195 PTAS.
(IVA Incluido)



CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

P.V.P. Canarias, Ceuta y Melilla 185 Ptas.

Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen X-Fascículo 118

Director:	José Mas Godayol
Director editorial:	Gerardo Romero
Jefe de redacción:	Pablo Parra
Coordinación editorial:	Jaime Mardones Francisco Martín
Asesor técnico:	Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 - Télex: 93392 EPPA

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 120 fascículos de aparición semanal, encuadernables en diez volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-183-6 (tomo 10)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 228604
Impreso en España-Printed in Spain- Abril 1986

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (120 fascículos más las tapas, guardas y transferibles para la confección de los 10 volúmenes) son las siguientes:

a) Un pago único anticipado de 27 105 ptas. o bien 10 pagos trimestrales anticipados y consecutivos de 2 711 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

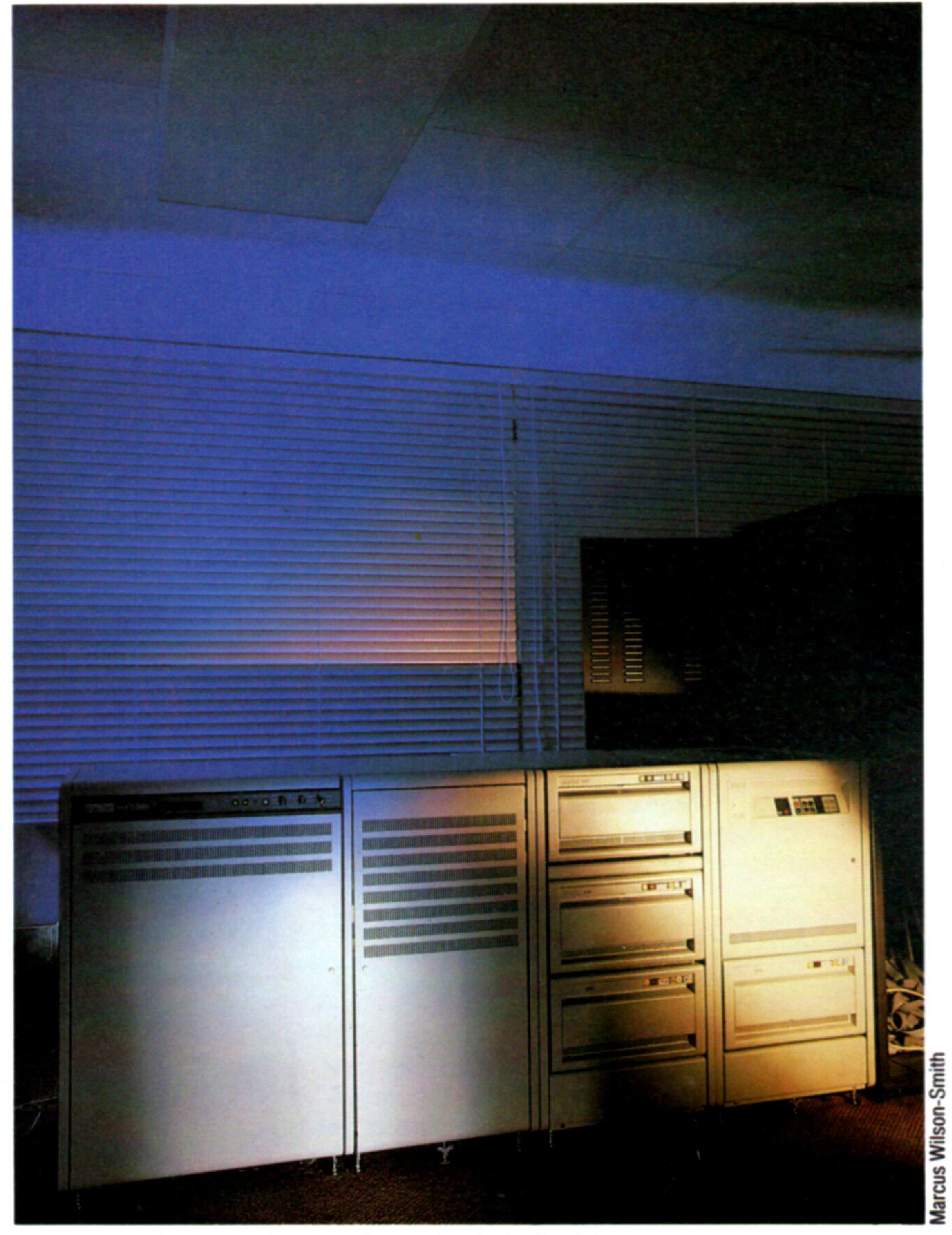
No se efectúan envíos contra reembolso.





Potencia inigualada

El miniordenador DEC VAX 11/780 es un excelente representante de las potentes máquinas de bases de datos y desarrollo de hoy en día



Central de fuerza de proceso
Por su alto precio, el miniordenador DEC VAX 11/780 es una opción viable solamente para empresas medianas y grandes. No obstante, por lo general se considera que los beneficios derivados del almacenamiento en línea masivo, la elevada velocidad de proceso de datos y la flexibilidad de esta facilidad de cálculo integrada bien valen los cuantiosos gastos de adquisición y mantenimiento

Dadas la potencia y velocidad de los micros actuales y el advenimiento de la conexión en red de software y hardware, sería lícito preguntarse por qué la gente continúa comprando miniordenadores muy caros de fabricantes como Prime y DEC. ¿Acaso estos mismos clientes no podrían reemplazar estas máquinas por ordenadores IBM PC y un par de discos para cada uno? La razón fundamental para adquirir un miniordenador, sin embargo, reside en su potencia bruta. La potencia de un ordenador se puede juzgar considerando su velocidad de cálculo, dimensiones de memoria y la gama de software y firmware disponible. En este sentido, ninguno de los micros actuales se aproxima a la potencia de los minis de tamaño mediano, e incluso en una red la potencia de cálculo está localizada: cada usuario está limitado a la potencia de su propia máquina.

Además, si un usuario está leyendo un listado, entonces su CPU estará ociosa, impidiendo que alguien de la red pueda acceder a ella. Frente a esta situación, la potencia completa de un miniordenador está disponible para todos los usuarios, si bien sólo parte del tiempo, y si un usuario no necesita su "parte de tiempo", entonces la misma será aprovechada por los otros usuarios del sistema.

Uno de los miniordenadores que ha tenido mayor aceptación es el VAX, de Digital Equipment Corporation (DEC). El VAX, de hecho, viene en varios modelos, que van desde el MicroVax, que al menos en tamaño es comparable a un micro de gestión estándar, hasta el superior de la gama, el 11/785. Nosotros hemos utilizado el 11/780, aunque la arquitectura es la misma para toda la gama, exceptuando el 11/782 y el 11/785.

Físicamente, el VAX es bastante diferente a los micros de gestión o personales estándares. En primer lugar, es más grande: el mueble básico mide 150 cm de altura, 100 cm de anchura y 75 cm de profundidad. Asimismo, mientras que la mayoría de los micros personales están incorporados en una única placa, utilizándose placas extras sólo con fines de ampliación, un 11/780 puede acomodar 16 placas que contengan solamente chips de memoria, si bien siempre contendrá otras varias placas que incluyan hardware para interface y la CPU.

Una CPU VAX no se halla en un único chip, ni siquiera en una única placa. Está diseñada para ser multitareas y para permitir una manipulación eficaz de la "memoria virtual". El VAX es una máquina de 32 bits, lo que permite direccionar más de cuatro gigabytes (cuatro billones de bytes) de memoria, aunque, incluso con el reducido precio de la RAM, toda esa memoria sería aún muy costosa. El concepto de memoria virtual se desarrolló para sacar partido de las capacidades de direccionamiento de la máquina. Sólo una fracción de la memoria disponible es verdaderamente RAM, y el hardware de la máquina se utiliza para comprobar si la dirección requerida se halla actualmente en la memoria; de no ser así, se la busca automáticamente en disco.

Este proceso se conoce como *paginación*. La memoria se manipula en páginas de 512 bytes, y cuando la memoria está llena, las páginas que han experimentado modificaciones mientras se hallan en la memoria se vuelven a escribir en disco a medida que se van requiriendo nuevas páginas. Todo esto, es completamente transparente para el usuario, dado que todos los programas "creen" estar hablándoles a cuatro gigabytes completos. En realidad, el hardware hace todo el trabajo de forma automática.

El sistema operativo estándar de DEC es VMS (aunque está disponible el Unix), que posee un potente lenguaje de comandos y un amplio juego de utilidades, incluyendo, entre otras, editores de línea y de pantalla completa, rutinas de clasificación y facilidades para correo electrónico. El único lenguaje proporcionado con la máquina es un macroensamblador, pero tanto DEC como otras fuen-

tes ofrecen compiladores para casi cualquier lenguaje por un costo adicional. El assembly VAX es una especie de revelación para los programadores habituados al código Z80 o 6502. Hay, por ejemplo, 16 registros, cada uno de los cuales es de 32 bits en tamaño. Cuatro de los mismos son especiales, utilizándose como punteros de pila y contador del programa, pero hay 12 para uso general. Y mientras los micros por lo general sólo pueden manipular en hardware aritmética de enteros, el VAX posee instrucciones para punto flotante (hasta números de punto flotante de 128 bits) y manipulación de caracteres. Asimismo, hay una única instrucción que corresponde a la estructura de bucle FOR...NEXT del BASIC. El juego total de 248 instrucciones incluye las operaciones básicas de mover y comparar, y un conjunto de instrucciones especiales para control de la CPU.

Para usar un VAX usted necesita un nombre de usuario y contraseña, asignados por el director del sistema, que le permite entrar en el sistema (*login*). Una vez en la máquina, se tiene acceso a las facilidades, cualesquiera que sean, que el director del sistema haya decidido que necesita el usuario. Cada nombre de usuario tiene privilegios y derechos asociados que limitan los archivos a los cuales se puede acceder, y también impide que usuarios no autorizados lleguen a ciertas partes del sistema operativo. Además, es posible proteger los propios archivos de modo que sólo ciertas personas puedan escribir en ellos, leerlos, suprimirlos o modificarlos.

El lenguaje de comandos se denomina DCL (Digital Command Language) e incluye muchas facilidades que normalmente sólo se encuentran en lenguajes de alto nivel. Éstas incluyen variables con nombre, sentencias GOTO e IF similares a sus equivalentes de BASIC, y un conjunto de funciones *lexicográficas* que se utilizan para devolver información del sistema tal como la hora y los nombres de los usuarios que están usando el sistema. La mayoría de estas funciones en realidad sólo se utilizan en *procedimientos de comandos*, que básicamente son listas de comandos DCL almacenados en un archivo y ejecutados como un programa. Para uso normal, no obstante, se emplea un único comando: por ejemplo, Show Users imprimiría una lista de usuarios. Una función lexicográfica se podría utilizar dentro de un procedimiento de comando, por ejemplo, para enviar un mensaje a cada usuario.

La mayoría de los comandos poseen varios cualificadores que se utilizan para modificar la acción. A modo de ejemplo, DELETE*.* suprimirá todos los archivos de un directorio (sólo aquellos que no estén protegidos) y DELETE/CONFIRM*.* llevará a cabo la misma acción, pero al usuario se le avisará en cada archivo para su confirmación. La lista de comandos completa es exhaustiva, e incluye todos los comandos usuales para tratamiento de archivos, así como algunos para modificar el entorno de trabajo y para permitir comunicarse con otros usuarios.

La comunicación con la máquina se efectúa a través de dos interfaces especiales. El Unibus se utiliza para terminales, discos lentos, unidades de cinta, impresoras y diversos equipos para fines especiales, como trazadores y dispositivos de control. La otra interface es Massbus, que es mucho más veloz y se utiliza para unidades de disco de acceso rápido.

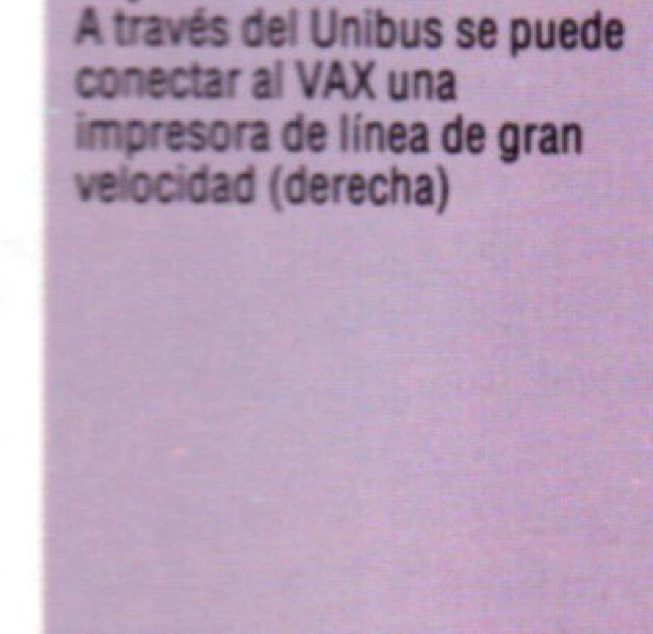
La configuración más pequeña de un 11/780 posee dos megabytes de memoria y ocho líneas para terminales. Adquiera al menos dos unidades de disco, cada una de las cuales debe ser capaz de almacenar al menos dos megabytes en un pequeño sistema operable. Se pueden instalar hasta ocho unidades de disco, y ampliar la memoria a ocho megabytes. Es posible conectar hasta 64 terminales con sólo instalar placas de interface adicionales.

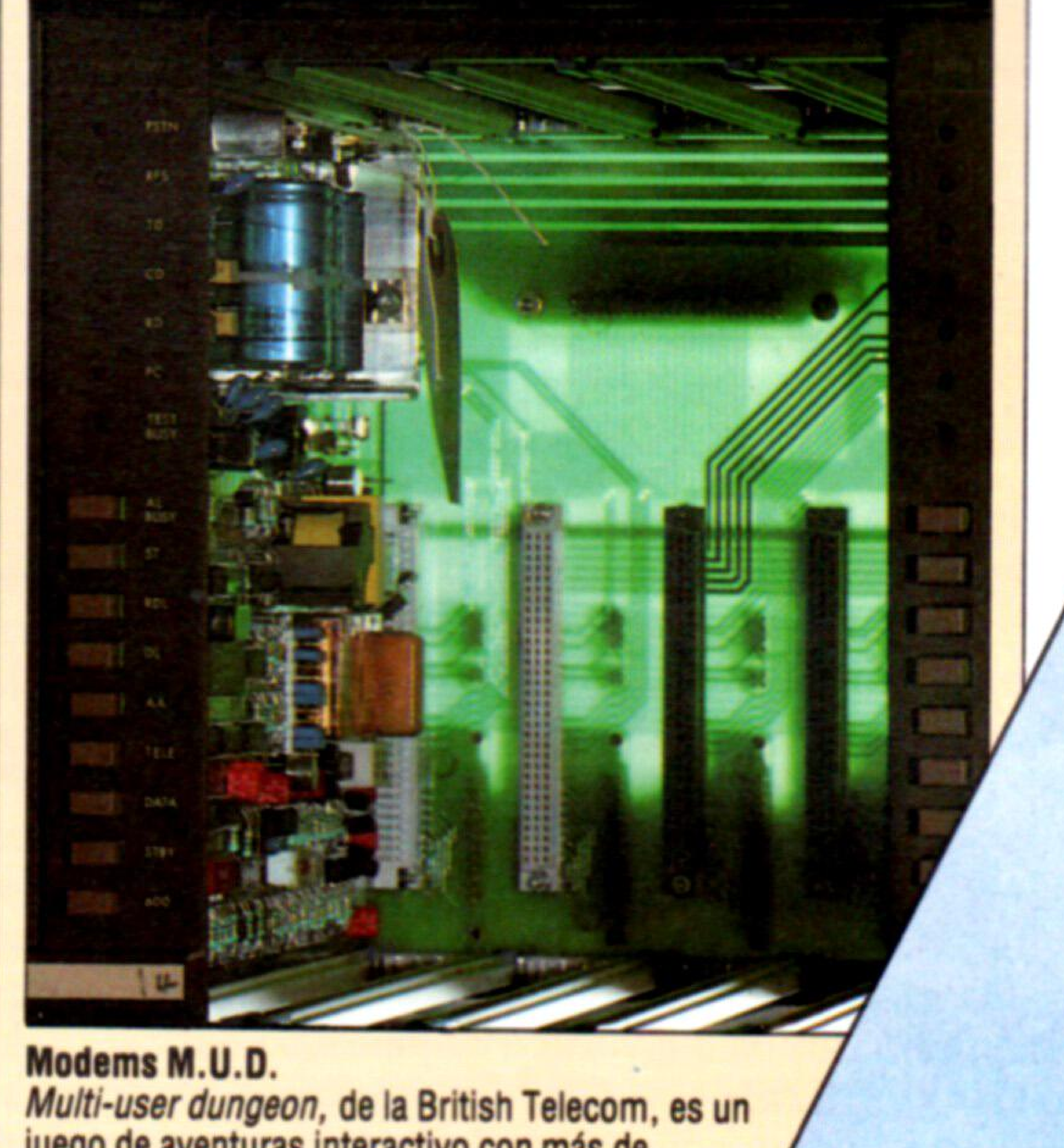
Como entorno de desarrollo el VMS es excelente: el lenguaje de comandos es lógico, y hay archivos de texto de ayuda en línea para todos los comandos y cualificadores de comandos. También se proporcionan bibliotecas de útiles rutinas para usar con sus propios programas. Ello significa que el desarrollo de tareas tales como manipulación de pantalla y rutinas matemáticas con fines especiales se elimina, lo que da por resultado una gran reducción del tiempo de programación. Los archivos en disco compartidos por un equipo de desarrollo eliminan los programas derivados de los sistemas de micros independientes cuando alguien cambia un archivo sin decir a los otros usuarios que cambien los suyos.

El miniordenador proporciona un entorno de programación muy amable, y la potencia de la máquina permite que los programas de aplicaciones complejas se ejecuten más rápidamente que en un micro; pero la potencia es cara. Asimismo, los miniordenadores generan elevados costos de mantenimiento, de electricidad (alrededor de 6 kW como mínimo) y precisan aire acondicionado. Sin embargo, a medida que los micros vayan adquiriendo mayor potencia, reemplazarán paulatinamente a los miniordenadores; no obstante, éstos proporcionan a los usuarios un rendimiento que sólo es superado por el de los ordenadores centrales.

La consola norteamericana
El director del sistema del VAX se comunica con la máquina a través de la consola del operador (derecha), utilizándola casi como si se tratara de una telemáquina de escribir. El ordenador también puede enviar respuestas y mensajes de estado a la consola, donde se imprimen. De este modo se produce una salida impresa completa de cualquier diálogo entre el director y la máquina

Impresora de línea
A través del Unibus se puede conectar al VAX una impresora de línea de gran velocidad (derecha)





Marcus Wilson-Smith

Modems M.U.D.
Multi-user dungeon, de la British Telecom, es un juego de aventuras interactivo con más de 1000 escenarios, para participar en él los usuarios deben llamar desde micros equipados con modems. El juego se ejecuta durante las horas no laborables en un miniordenador VAX, que por el día actúa como una base de datos europea. El VAX está equipado con numerosos modems (como vemos arriba) conectados a su red Unibus, permitiendo que hasta 100 personas jueguen simultáneamente, cada una a través de su propio enlace telefónico







Kevin Jones



DEC VAX 11/780

DIMENSIONES
1 181 × 762 × 1 537 mm

CPU
VAX CPU

MEMORIA
RAM de entre 2 y 8 megabytes. La gestión de memoria virtual permite acceder a 4 gigabytes

PANTALLA
Visualización de textos de al menos 80 × 25, y resolución de hasta 1 024 × 1 024 pixels, con 16,5 millones de colores (según el tipo de terminal)

INTERFACES
Massbus para discos y cintas rápidos, Unibus para todos los otros dispositivos, incluyendo impresoras, trazadores y terminales, a través de un enlace RS232

LENGUAJES DISPONIBLES
Proporcionado con kit ensamblador, pero a un costo adicional hay disponibles compiladores para la mayoría de los lenguajes de alto nivel

DOCUMENTACION
El conjunto básico de documentación para el lenguaje de comandos se eleva a 30 volúmenes, ocupando 1,5 m de espacio de estantería. Éste incluye una concisa sección de referencia y guías de formación

VENTAJAS
El VAX proporciona un entorno de programación sumamente potente para desarrollo de software y administración de bases de datos. También es posible la conexión en red entre máquinas VAX

DESVENTAJAS
El sistema es bastante caro e implica elevados costos de operación, puesto que requiere una atmósfera libre de polvo y mantenimiento y apoyo a tiempo completo

Minisistema
El VAX se comunica con sus periféricos a través de dos buses. Massbus es un bus de gran velocidad que permite la rápida transferencia de datos entre máquinas de disco o de cinta de gran velocidad y el ordenador. Unibus se utiliza para los periféricos más lentos. Se pueden instalar ocho terminales, así como impresoras y trazadores. El Unibus le añade flexibilidad al sistema VAX, puesto que permite la conexión al mismo de enlaces de comunicaciones y dispositivos de control de procesos industriales

Escriba su propio guión

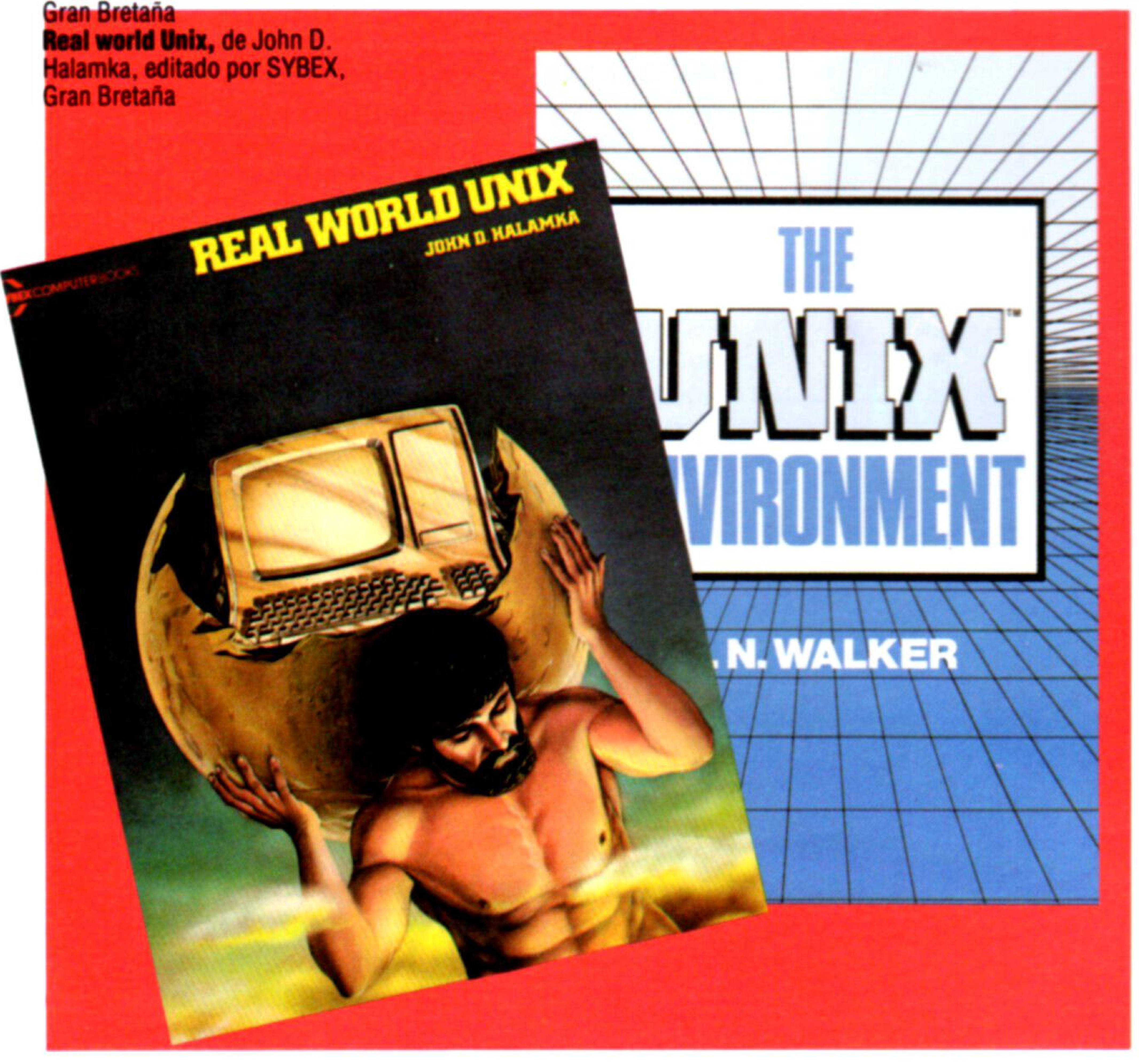
Presentamos un ejemplo que ilustra significativamente la adaptabilidad del Unix a las exigencias del usuario

Textos sobre Unix
Es posible encontrar algunos interesantes libros que tratan del Unix. Entre éstos se incluyen tanto obras de "referencia" rápida para el programador experto como sencillas guías de introducción para el usuario ocasional. *The Unix environment* (El entorno Unix), de A. N. Walker, está escrito en un estilo desenfadado y es indicado para aquellos lectores que se inician en el Unix. Cubre las principales facilidades del sistema operativo, y también analiza la filosofía del Unix, sus facilidades típicas de hardware y la administración del sistema.

Real world Unix (El mundo real del Unix), de John D. Halamka, constituye un excelente texto práctico para quienes deseen aprender el Unix en el menor tiempo posible. El texto contiene numerosos ejemplos de las tareas más comunes y una guía de referencia rápida "el Unix en un minuto".

The Unix environment, de A. N. Walker, editado por John Wiley, Gran Bretaña

Real world Unix, de John D. Halamka, editado por SYBEX, Gran Bretaña



La mayoría de los sistemas operativos incorporan facilidades que permiten que los usuarios adapten el sistema para satisfacer sus necesidades específicas. Generalmente incluyen las opciones de ejecutar un lote de comandos a la vez (los archivos SUBMIT de CP/M y .BAT del MS-DOS, p. ej.) y ejecutar una determinada secuencia de comandos o programas cuando el usuario conecta por primera vez el sistema. El Unix, sin embargo, va más allá.

El Unix posee un "guión de caparazón", un lenguaje de programación por derecho propio, que posee facilidades para entrada/salida, selecciones e iteraciones. En combinación con los operadores de redirección y entubamiento del Unix, permite que los usuarios adiestrados construyan programas para la mayoría de las tareas sin necesidad de un lenguaje de programación "convencional".

El caparazón es el equivalente del procesador de comandos del CP/M y el usuario puede reescribirlo por completo. Existe un ligero inconveniente por el hecho de que distintos sistemas Unix puedan presentar al usuario distintos "rostros", aunque el sistema subyacente sea idéntico. Hay dos caparazones comunes para el sistema Unix Berkeley, en el cual se basa esta serie. El "caparazón c", como su nombre sugiere, se basa en el lenguaje c y es el que hemos utilizado para nuestros ejemplos. Sin embargo, si hubiéramos empleado el "caparazón Bourne", habría habido muy pocas diferencias en lo que hemos considerado hasta ahora.

Veamos ahora algunos de los comandos que afectan el entorno del usuario. Ya hemos visto brevemente cómo cada archivo o directorio puede acceder a privilegios para tres grupos de personas: los usuarios propiamente dichos (u), el grupo del usuario (g) y otros (o). (Esta idea de grupos es muy útil, especialmente en grandes organizaciones en las que es vital compartir los recursos y evaluar cada área de trabajo.) A cada una de estas tres categorías se le otorgan derechos de acceso para cada archivo y directorio del sistema, denotados mediante tres juegos de rwx en el listado de directorio completo dado por el comando ls–1. La r denota acceso para leer, la w para escribir y la x para ejecutar. Si una categoría no goza del acceso apropiado, la letra se sustituye por un guión.

Estos derechos de acceso se pueden cambiar utilizando la instrucción chmod, que toma la forma:

```
chmod categoría [+,–] acceso nombrearchivoodirectorio
```

donde + se utiliza para dar un derecho de acceso, y –, para suprimirlo. Así que, para otorgar al usuario y a su grupo derecho de escritura a un archivo llamado trabajoparcial, sería aceptable lo siguiente:

```
chmod ug+w trabajoparcial
```

Vea que se puede dar más de un cambio de permiso, separado por comas (pero no por espacios).

Otra opción permite cambiar los parámetros para el determinado tipo de terminal que se esté utilizando. La mayoría de los sistemas Unix vienen con tablas incorporadas para toda una variedad de tipos de terminal, uno de los cuales se puede seleccionar mediante el empleo de:

```
setenv TERM tipoterminal
```

Esto significa que no es necesario instalar paquetes de software para una pantalla y teclados determinados; en teoría, una instalación funcionará para todos ellos.

Para los tipos de terminal no estándares y para determinados fines, se pueden cambiar las teclas utilizadas para varias funciones con la instrucción stty:

```
stty everything
```

que dará una lista de los ajustes disponibles, y:

```
stty all
```

que dará una lista restringida de los más comunes.

Tarea a medida

```
Berkeley 4.2 Vax/Unix (infsc3)
Type (Ctrl-D) to disconnect

login: com–mcc
Password:
You are a Normal user (class 3)
Jobs : 15 Superiors : 4 Maximum : 21
Last Login: Wed Nov 13 15:05:36 on ttyn08

Hello nice to see you!          {éste es mi mensaje login de usuario}

%cat .login                     {consultar en archivo login}
setenv TERM tvi910+             {establecer mi tipo de terminal}
stty erase                      {Ctrl-H me permite usar el retroceso en mi PC en
                                lugar de la tecla 'del'}
echo Hello nice to see you!     {éste es mi mensaje de sign-on}

%cat .cshrc
setenv EDITOR /usr/local/emacs
setenv NAME 'Mike Curtis'
set paht=(./usr/local/m2/usr/local/usr/ucb/usr/bin/bin/usr/games)
{el camino establece los directorios que se buscarán (por el orden dado)
para comandos, de modo que, si fuera necesario, se podría acceder directamente
a programas de otros directorios. El primer '.' es el directorio del propio
usuario}

set history=25                  {el caparazón recordará los últimos
                                25 comandos}
set ignnoreeof                  {detiene desconexión accid. con Ctrl-D}
alias h history                 {adaptar comandos Unix a medida}
alias ty more
alias dir ls                    {usar dir en lugar de ls para listar
                                el directorio}

%dir                            {demostrar instrucción dir}
lsfile mike rec.c receive rx.p transmit

%cat > ! temp                   {cat sin un archivo de entrada tomará la entrada
                                desde el teclado hasta un Ctrl-D}
```

```
echo this is a list of mu files
/bin/ls                         {el n. completo del comando ls}
%mv temp ls                     {elaborar el propio comando ls a la medida}

%chmod u+x ls                   {usar chmod para hacerlo ejecutable. De lo
                                contrario, ejecutado mediante sh < ls}
%ls                             {probar versión hecha a medida}
this is a list of my files
ls lsfile mike rec.c receive rx.p transmit

%history                        {repasar comandos impartidos}
    1 dir
    2 cat < ! temp
    3 mv temp ls
    4 chmod u+x ls
    5 ls
    6 history

% !5                            {repetir comando 5}
ls                              {reproduce primero el comando repetido}
this is a list of my files
ls isfile mike rec.c rewceive rx.p transmit

%stty everything                {ver todos los ajustes de terminal}
new tty, spped 1200 baud
even odd –raw –nl echo –lcase –tandem –tabs –cbreak ff1
–crtbs –crterase –crtkill –ctlecho –prterase –tostop
–tilde –flusho –mdmbuf –litout – nohang
–pendin –decctlq –noflsh
erase kill werase rprnt flush lnext susp intr quit stop eof
^H    ^U   ^W      ^R      ^O  ^V  ^Z/^Y  ^C  ^    ^S/^Q  ^D

%stty all                       {algunos ajustes de teclas comunes}
new tty, speed 1200 baud; –tabs ff1

erase kill werase rprnt flush lnext susp intr quit stop eof
^H    ^U   ^W      ^R      ^O  ^V  ^Z/^Y ^C  ^    ^S/^Q  ^D

%logout
```

Cada función posee un nombre y se puede cambiar:

stty nombrefunción nuevatecla

Otra facilidad útil impide el uso sin autorización del terminal mediante el empleo de lock. Este comando solicitará una contraseña (distinta de la contraseña de login) y luego pedirá que se la repita. Entonces el terminal estará bloqueado y el Unix ignorará todas las pulsaciones de tecla hasta que se entre la contraseña una tercera vez.

Las dos facilidades siguientes constituyen una peculiaridad del caparazón c, pero son a la vez interesantes e instructivas. El comando history dará una lista de todos los comandos impartidos previamente, hasta una cantidad predefinida. Además, cualquiera de los comandos se puede repetir utilizando el operador !, por lo general seguido por el número de la lista del comando a repetir. De modo que:

!4

haría que se repitiera el cuarto comando de la lista.

El comando alias se puede utilizar para asignar otro nombre ya sea a cualquier comando o a cualquier serie de caracteres. Cada vez que se utilice el nuevo nombre, el Unix sustituirá la serie en su sitio. Para los usuarios de CP/M o MS-DOS que quieran utilizar dir para obtener un listado de directorio en lugar del comando ls del Unix:

alias dir ls

aceptará dir como ls, que puede entonces ir seguido por cualquiera de las opciones usuales. Si la serie contiene un punto y coma, que el Unix utiliza para separar comandos en una línea, se puede encerrar entre comillas:

alias dir 'fechas;ls -a'

El uso juicioso de alias puede aliviar muchas de las características hostiles del Unix.

En la mayoría de los directorios de los usuarios existen numerosos archivos especiales del sistema que se distinguen por tener nombres que comienzan con un punto. Éstos se protegen contra la supresión accidental mediante caracteres de máscara que no posean un punto delante. Dos de estos archivos son de especial interés, dado que preparan el entorno para un usuario: (.login y .cshrc [en el caparazón Bourne, sus funciones las cubre un único archivo, .profile]) y cada uno contiene un número de comandos de caparazón. El archivo .login se ejecuta cada vez que el usuario entra en el sistema, y se remite a .cshrc cada vez que se activa el programa de caparazón (estos archivos por lo general contienen fundamentalmente comandos alias, setenv y set). Las posibles opciones para setenv y set dependen de la instalación individual y son demasiado numerosas como para detallarlas aquí. Pero basta decir que se puede establecer virtualmente cualquier aspecto del sistema y los dispositivos periféricos según las propias especificaciones del usuario.

Los toques finales

En este penúltimo capítulo de nuestra serie presentamos la rutina de compresión en assembly 6502 y un programa Cargador en BASIC para la versión Z80 que ofrecimos en el capítulo anterior. La segunda mitad del programa y los programas Cargador y Activador en BASIC los incluiremos en el capítulo final

Rutina de compresión 6502

```
;++++ COMPRESION DE TEXTOS 6502 ++++
;
ZPTR1=$8B                       ;PUNTERO ENTRADA PAGINA 0
ZPTR2=$8D                       ;PUNTERO SALIDA PAGINA 0
ZPTR3=$FB                       ;PUNTERO TABLA UTILIDADES
ZPTR4=$FE                       ;PUNTERO TABLA UTILIDADES
*=$C000
OUTPUT   *=*+2                  ;COMIENZO DE SERIE O/P
INPUT    *=*+2                  ;COMIENZO DE SERIE I/P
STATUS   *=*+1                  ;BYTE DE ESTADO
MASK     *=*+1                  ;MASCARA NIBBLE
LEN      *=*+1                  ;ENTRADA ALMACENAMIENTO LONG.
                                SERIE
OUTOFF   *=*+1                  ;SALIDA ALMACENAMIENTO DESPL.
TOKCNT   *=*+1                  ;CONTADOR DE DISTINTIVOS
TABCNT   *=*+1                  ;CONTADOR TABLA
TEMP     *=*+1                  ;GUARDAR COMIENZO DE SERIE O/P
;
START    LDA  INPUT
         STA  ZPTR1             ;ESTABLECER PUNTERO I/P
         LDA  INPUT+1
         STA  ZPTR1+1
         LDA  OUTPUT
         STA  ZPTR2             ;ESTABLECER PUNTERO O/P
         LDA  OUTPUT+1
         STA  ZTR2+1
         LDA  #1
         STA  OUTOFF            ;INIC. DESPL. O/P
         LDY  #0                ;INIC. DESPL. I/P
         LDA  (ZPTR1), Y
         STA  LEN               ;ALMACENAR LONGITUD SERIE I/P
         LDA  #255
         STA  MASK              ;ESTABLECER MASCARA NIBBLE
NCHAR    INY
         LDA  (ZPTR1), Y        ;TOMAR SIGUIENTE CAR I/P
         JSR  CHECK             ;VALIDO?
         BNE  BADCHR
         JSR  TOKEN             ;DISTINTIVO?
         BEQ  GOTTOK
         JSR  FOURBT            ;CODIGO CUATRO BITS?
         BEQ  GOTFOR
         JSR  EIGHTB            ;CODIGO OCHO BITS?
         PHA                    ;GUARDAR VALOR
         LDA  #0
         JSR  WRTNIB            ;ENVIAR CODIGO
         PLA
GOTFOR   JSR  WRTNIB            ;ENVIAR CODIGO 8 BITS
;
REJOIN   CPY  LEN               ;MAS CARACTERES I/P?
         BCC  NCHAR             ;IR POR ELLOS
         LDA  #0
         STA  STATUS            ;ESTADO 0=OK
         LDA  #2
         JSR  WRTNIB            ;FIN DE MARCA TEXTO
         LDA  MASK
         BEQ  NODEC
         DEC  OUTOFF
NODEC    LDY  #0
         LDA  OUTOFF
         STA  (ZPTR2),Y         ;ESCRIBIR LONGITUD O/P
         RTS
;
BADCHR   LDA  #255
         STA  STATUS            ;ENVIAR ESTADO MALO
         RTS
GOTTOK   PHA
         LDA  #1
         JSR  WRTNIB            ;ENVIAR CODIGO 0001
         PLA
         JSR  WRTNIB            ;Y CODIGO DISTINTIVO
         JMP  REJOIN
;
;++++    SUBRUTINAS+++++
TOKEN    PHA                    ;GUARDAR CAR. ACTUAL
         TYA
         PHA                    ;Y DESPL. I/P
         STY  TEMP
         LDA  ZPTR1
         CLC
         ADC  TEMP              ;CALC ZPTR3 PARA
         STA  ZPTR3
         LDA  ZPTR1+1           ;APUNTAR AL ACTUAL
         ADC  #0                ;CAR. I/P
         STA  ZPTR3+1
         LDA  #<TOKTAB
         STA  ZPTR4             ;ESTABLECER ZPTR4
         LDA  #>TOKTAB          ;PARA APUNTAR A
         STA  ZPTR4+1           ;TABLA DISTINTIVOS
         LDA  #0
         STA  TOKCNT            ;INIC. CONTADOR DISTINTIVOS
TOK1     LDY  #0
         LDA  (ZPTR4),Y         ;TOMAR LONGITUD SIGUIENTE
                                DISTINTIVO
         PHA                    ;ALMACENARLA
         TAX
         BEQ  NOTFND            ;FIN DE LA TABLA
         LDA  ZPTR4
         CLC
         ADC  #1                ;INC ZPTR4
         STA  ZPTR4             ;ZPTR3 & ZPTR4
         LDA  ZPTR4+1           ;AMBOS AHORA EN
         ADC  #0                ;COMIENZOS TEXTO
         STA  ZPTR4+1
TOK2     LDA  (ZPTR3),Y
         CMP  (ZPTR4),Y         ;COMPARAR
         BNE  NEXTOK            ;NINGUNA PAREJA TOMAR SIGUIENTE
                                DISTINTIVO
         INY
         DEX                    ;CONTHASTA
         BNE  TOK2              ;FIN PALABRA DISTINTIVO
;++ DISTINTIVO EMPAREJADO ++
         PLA                    ;LIMPIAR PILA
         PLA
         DEY
         STY  TEMP
         CLC
         ADC  TEMP              ;CALC NUEVO DESPL. I/P
         TAY                    ;PONERLO EN Y
         PLA
         LDA  TOKCNT
         LDX  #0                ;ESTABLECER Z=1
         RTS
;
NEXTOK   PLA                    ;TOMAR LONGITUD DISTINTIVO
         STA  TEMP
         LDA  ZPTR4
         CLC
         ADC  TEMP
         STA  ZPTR4             ;AJUSTAR ZPTR4
         LDA  ZPTR4+1           ;PARA QUE APUNTE AL
         ADC  #0                ;SIGUIENTE DISTINTIVO
         STA  ZPTR4+1
         INC  TOKCNT
```



Compresión veloz
El uso de distintivos, más la representación de caracteres comunes en un código de cuatro bits, permite que nuestro algoritmo de compresión de textos consiga velocidades de compresión que alcanzan el 50 %. Aquí, una serie de ocho letras (incluyendo el espacio) se reduce a cuatro bytes. Observe los códigos de control de 4 bits que indican una palabra distintivada, un segundo juego de caracteres y el fin del mensaje

ANTES

1	2	3	4	5	6	7	8
G	E	T		T	H	I	S

DESPUES

0000	G	E	T	ESPACIO	0001	THIS	0010
1		2		3		4	

```
        JMP  TOK1
NOTFND  PLA               ;DESCARTAR
        PLA
        TAY               ;RESTAURAR PUNTERO I/P
        PLA               ;RESTAURAR CAR. I/P
        LDX  #255         ;Z=0=FALLO
        RTS
;
FOURBT  PHA               ;GUARDAR CAR. I/P
        STY  TEMP         ;GUARDAR DESPL. I/P
        LDA  #<TAB4BT
        STA  ZPTR3
        LDA  #>TAB4BT
        STA  ZPTR3+1
        PLA               ;VOLVER A TOMAR CAR I/P
        JMP  TBSCAN
EIGHTB  PHA               ;GUARDAR CAR I/P
        STY  TEMP         ;GUARDAR DESPL. I/P
        LDA  #<TAB8BT
        STA  ZPTR3
        LDA  #>TAB8BT
        STA  ZPTR3+1
        PLA               ;VOLVER A TOMAR CAR I/P
;
TBSCAN  LDY  #0
TBSCN2  CMP  (ZPTR3),Y
        BEQ  TBSCN1       ;HALLADO!
        INY
        CPY  #16          ;BUSCAR FIN TAB
        BNE  TBSCN2
        LDY  TEMP         ;RESTAURAR DESPL. I/P
        LDX  #255         ;Z=0=FALLO
        RTS
TBSCN1  TYA               ;PONER VALOR TAB. EN A
        LDY  TEMP
        LDX  #0           ;Z=1=EXITO
        RTS
;
CHECK   CMP  #' '
        BEQ  RETURN
        CMP  #','
        BEQ  RETURN
        CMP  #'.'
        BEQ  RETURN
        CMP  #'A'
        BCC  NOGOOD
        CMP  #$5B
        BCS  NOGOOD
RETURN  LDX  #0
        RTS
NOGOOD  LDX  #255
        RTS
;
WRTNIB  PHA               ;GUARDAR NIBBLE
        STY  TEMP         ;GUARDAR DESPL. I/P
        LDY  OUTOFF
        LDA  MASK
        BNE  LEFT         ;<> 0 SIGNIFICA LADO IZQUIERDO
        PLA
        ORA  (ZPTR2),Y    ;AÑADIR EL NUEVO AL ANTIGUO
        STA  (ZPTR2),Y    ;SUSTITUIRLO
        INC  OUTOFF
        LDY  TEMP         ;RESTAURAR DESPL. I/P
        LDA  #255
        STA  MASK         ;RESTABLECER MASCARA PARA
                          SIGUIENTE
        RTS
LEFT    PLA
        ASL  A
        ASL  A
        ASL  A
        ASL  A
        STA  (ZPTR2),Y
        LDY  TEMP
        LDA  #0
        STA  MASK
        RTS
;
;++++ LETRAS MAS COMUNES ++++
TAB4BT  .BYT  0
        .BYT  0
        .BYT  0
        .BYT  'F'
        .BYT  'L'
        .BYT  'D'
        .BYT  'H'
        .BYT  'S'
        .BYT  'I'
        .BYT  'R'
        .BYT  'N'
        .BYT  'O'
        .BYT  'A'
        .BYT  'T'
        .BYT  'E'
        .BYT  ' '
;++++ LETRAS MENOS COMUNES ++++
TAB8BT  .BYT  'C'
        .BYT  'M'
        .BYT  'U'
        .BYT  'G'
        .BYT  'Y'
        .BYT  'P'
        .BYT  'W'
        .BYT  'B'
        .BYT  'V'
        .BYT  'K'
        .BYT  'X'
        .BYT  'J'
        .BYT  'Q'
        .BYT  'Z'
        .BYT  ','
        .BYT  '.'
;++++ TABLA DE DISTINTIVOS ++++
TOKTAB  .BYT 3, 'THE'

        .BYT 4, 'THIS'

        .BYT 4, 'THAT'

        .BYT 2, 'IF'
```

```
.BYT 3, 'YOU'
.BYT 2, 'ME'
.BYT 3, 'WAS'
.BYT 2, 'HE'
.BYT 3, 'SHE'
.BYT 4, 'THEY'
.BYT 2, 'OF'
.BYT 2, 'IT'
.BYT 2, 'IS'
.BYT 3, 'FOR'
.BYT 2, 'ON'
.BYT 2, 'TO'
.BYT 0
.END
```

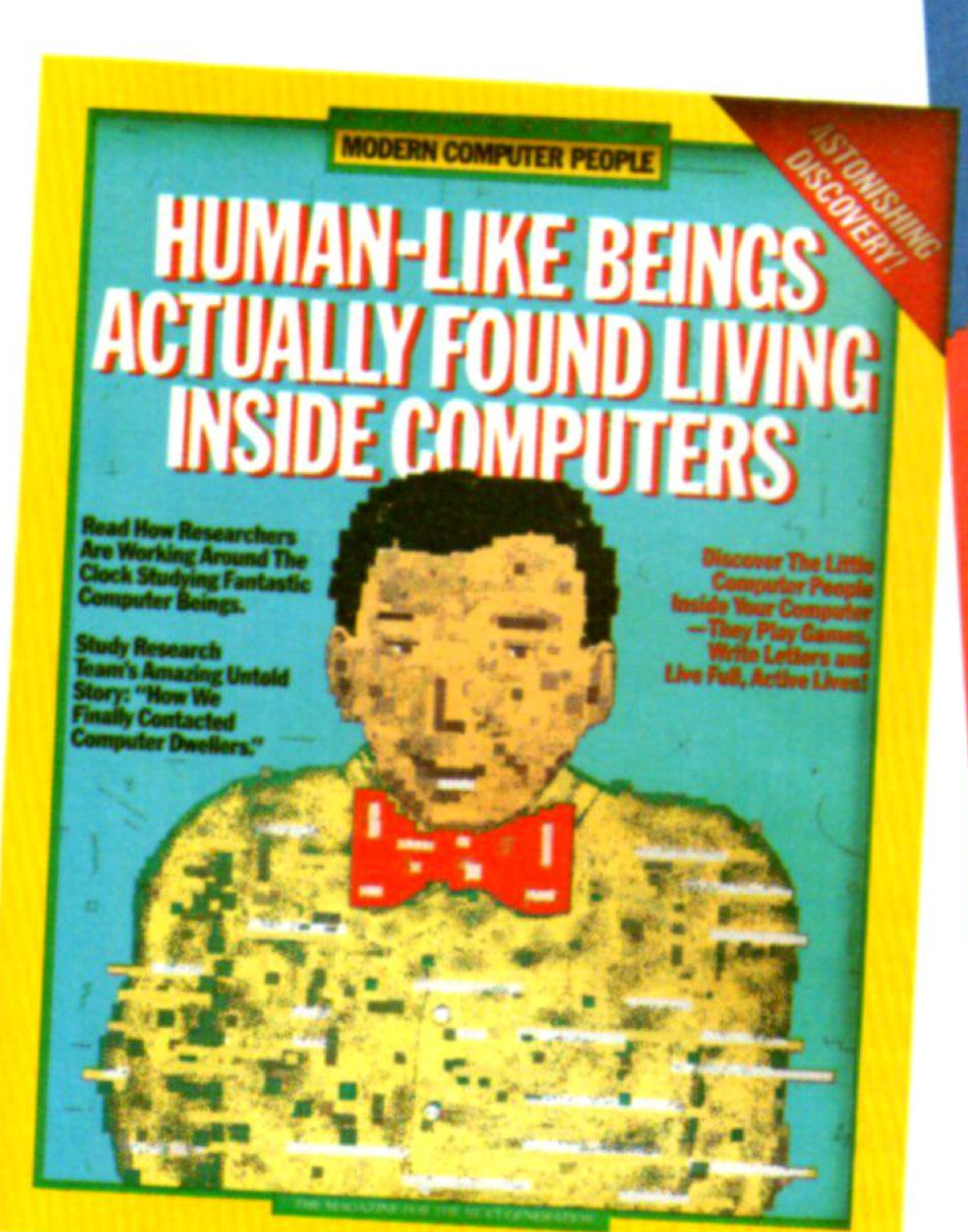
Cargador en BASIC Z80

```
10 MEMORY 29999
20 FOR n=30000 TO 30640
30 READ p:POKE n,p:c=(c+p)
40 NEXT n
50 IF c< >64282 THEN PRINT" Error en suma de
   control..."
60 STOP
100 DATA 24,7,70,160,102,158,0,255,0,42,50,
    117,126,50,56,177
110 DATA 35,237,91,52,117,213,19,237,83,52,
    117,62,255,50,55,117
120 DATA 126,205,253,117,32,57,205,162,117,
    48,59,205,223,117,40
130 DATA 10,205,232,117,245,62,0,205,22,
    118,241,205,22,118,35
140 DATA 58,56,117,61,50,56,117,167,32,216,
    50,54,117,62,2,205
150 DATA 22,118,235,209,58,55,117,167,40,1,
    43,167,237,82,125,18
160 DATA 201,209,62,255,50,54,117,201,245,
    62,1,205,22,118,241
170 DATA 205,22,118,24,203,229,235,33,2,
    119,58,56,117,79,6,15
180 DATA 126,167,40,42,213,229,35,197,71,
    26,190,32,21,35,19,5
190 DATA 32,13,121,50,56,117,193,225,225,
    225,235,43,175,120,201
200 DATA 13,32,231,193,5,225,126,95,22,0,
    35,25,209,24,210,225
210 DATA 126,183,201,229,33,226,118,205
    241,117,225,201,229,33
220 DATA 242,118,205,241,117,225,201,6,15,
    190,40,5,35,16,250,183
230 DATA 201,120,201,254,32,200,254,44,200,
    254,46,200,254,65,56
240 DATA 8,254,91,48,4,79,175,121,201,62,
    255,167,201,79,58,55
250 DATA 117,237,91,52,117,167,32,14,26,
    177,18,19,237,83,52,117
260 DATA 62,255,50,55,117,201,121,203,39,
    203,39,203,39,203,39
270 DATA 18,175,50,55,117,201,42,50,117,35,
    34,50,117,237,91,52
280 DATA 117,213,62,255,50,55,117,205,185,
    118,254,2,202,129,118
290 DATA 210,153,118,167,202,139,118,205,
    185,118,205,110,118,71
300 DATA 126,35,205,174,118,16,249,24,255,
    32,2,119,71,62,15,144
310 DATA 71,4,126,35,5,200,95,22,0,25,24,
    246,42,52,117,209,167
320 DATA 237,82,125,18,201,205,185,118,33,
    242,118,205,164,118,205
330 DATA 174,118,24,182,33,226,118,205,164,
    118,205,174,118,24,171
340 DATA 95,62,15,147,95,22,0,25,126,201,
    237,91,52,117,19,18
350 DATA 237,83,52,117,201,58,55,117,42,50,
    117,167,126,32,14,230
360 DATA 15,35,34,50,117,79,62,255,50,55,
    117,121,201,203,47,203
370 DATA 47,203,47,203,47,230,15,79,175,50,55,
    117,121,201,32,69
380 DATA 84,65,79,78,82,73,83,72,68,76,70,
    0,0,0,67,77,85,71
390 DATA 89,80,87,66,86,75,88,74,81,90,44,
    46,3,84,72,69,4,84
400 DATA 72,73,83,4,84,72,65,84,2,73,70,3,
    89,79,85,2,77,69
410 DATA 3,87,65,83,2,72,69,3,83,72,69,4,
    84,72,69,89,2,79,70
420 DATA 2,73,84,2,73,83,3,70,79,82,2,79,
    78,2,84,79,0,205,167
430 DATA 126,32,14,230,15,35,34,50,117,79,
    62,255,50,55,117,121
440 DATA 201,203,47,203,47,203,47,203,47,
    230,15,79,175,50,55,117
450 DATA 121,201,32,69,84,65,79,78,82,73,
    83,72,68,76,70,0,0
460 DATA 0,67,77,85,71,89,80,87,66,86,75,
    88,74,81,90,44,46,3
470 DATA 84,72,69,4,84,72,73,83,4,84,72,65,
    84,2,73,70,3,89
480 DATA 79,85,2,77,69,3,87,65,83,2,72,69,
    3,83,72,69,4,84,72
490 DATA 69,89,2,79,70,2,73,0,0
```

Complementos al BASIC

Los usuarios del Spectrum necesitarán introducir una ligera modificación en el programa Cargador. La rutina de compresión también debe trabajar con otras máquinas Z80, incluyendo el Memotech, el Amstrad y el Einstein. Quizá los usuarios de algunas de estas máquinas también deban alterar la línea 10 del programa Cargador (consulte su manual para obtener detalles):

```
10 CLEAR 29999
```





Dimension Graphics

Personas queridas

He aquí un insólito programa, en el que el jugador debe tratar de hacer feliz a una diminuta persona que vive dentro del ordenador

A pesar del hecho de que el software de juegos por ordenador se ha desarrollado considerablemente desde los primeros días de los micros personales, la mayoría de los juegos aún tienen un objetivo predeterminado. Ya sean recreativos, de aventuras o estratégicos, el usuario tendrá como misión salir airoso de los desafíos ideados por el programador.

El programa se ha escrito de modo tal que la persona que hay en cada copia es, en cierto grado, exclusiva y única. Todas ellas tienen nombres diferentes, se visten de modo distinto y tienen prioridades exclusivas en cuanto a su forma de "vivir".

Obviamente se han dedicado enormes esfuerzos en proporcionarle a *Little computer people* historias en las cuales uno pueda sentirse comprometido. El software viene empaquetado con una gran carpeta en la que se detalla el "descubrimiento" de esta pequeña gente, costumbres, preferencias y hábitat.

Cuando se carga el programa por primera vez, la pantalla visualiza el interior de una casa, que cuenta con todas las habitaciones que cabría esperar en una casa normal de tres plantas. También hay un armario empotrado en el dormitorio y a menudo las personas desaparecen en el interior del mismo durante unos minutos, si bien no se sabe muy bien qué es lo que hacen realmente allí dentro. Unos minutos después de cargado el juego, su protegido entrará en la casa e inspeccionará sus dependencias. Si las aprueba, desaparecerá durante unos instantes antes de reaparecer con todas sus pertenencias, incluyendo su perrito.

Para mantener feliz a esta persona, usted ha de proporcionarle comida, agua y, sobre todo, muchísima diversión. El éxito que obtenga en la satisfacción de estas necesidades podrá evaluarlo por la expresión del rostro de la personilla, que puede ser desde desdichada hasta muy feliz. Por ejemplo, si ésta luce un aire desdichado y verde, es porque no se está alimentando muy bien. El jugador debe tener presente que también es necesario dar de comer al perro.

Activision insiste en que el idioma que hablan estas personas no se ha descifrado todavía, de modo que toda la comunicación con ellas se realiza a través del teclado. De este modo, es posible asegurarse de que su protegido sea feliz sugiriendo que tiene comida, puede encender fuego o tocar el piano. Es posible obtener un barómetro exacto del estado de ánimo de la persona sugiriendo que le escriba una carta a usted.

Se ha prestado mucha atención al detalle en la programación de los gráficos y el sonido, que es de muy buena calidad. La música producida cuando la personilla toca el piano es una de las mejores que han emanado nunca del Commodore 64. Lo que es más, se han sincronizado las manos de la misma con las notas y los acordes.

Little computer people: Para el Commodore 64
Editado por: Activision Inc, Box 7287 Mountain View, California, Estados Unidos
Autores: Richard Gold, David Crane y Sam Nelson
Formato: Cassette o disco
Palancas de mando: No son necesarias

Un día en la vida
Quienes desarrollaron *Little computer people* han dedicado todos sus esfuerzos a que las criaturas que habitan en el interior del ordenador parezcan lo más "reales" posible. La documentación con formato de revista que se entrega con el programa y que suele acompañar a todos los juegos es una de las más cuidadas que se hayan realizado. La casa en la cual se desarrolla la acción cuenta con todas las comodidades que comúnmente hacen más grata la vida. Aunque, todo hay que decirlo, en el transcurso del programa no suceden demasiadas cosas, el dedicarse a cuidar a esta "personita" tiene un atractivo emocional que muy pocos juegos pueden proporcionar

Escribir juegos

Estrella del software casero
El programador británico Jeft Minter es muy conocido entre los aficionados a los juegos recreativos. Sus programas han tenido un enorme éxito en un mercado que es sumamente competitivo, y todos ellos los ha desarrollado en su casa. La situación de programadores como Minter, sin embargo, a la larga puede ser inestable. Los programadores comerciales pueden beneficiarse del intercambio de ideas y la formación derivada de trabajar en una gran empresa, ventajas que les son negadas a los programadores por cuenta propia



A la conquista de usuarios
Virgin's Games Centre, en la Oxford Street londinense, es el ejemplo típico de una nueva generación de comercios alentados por el aumento en popularidad de los juegos de mesa y el software para ordenador. Sin embargo, la mayor parte de los programas se continúa vendiendo a través de las grandes cadenas minoristas, tales como W. H. Smiths y Boots. Convencer a una cadena minorista de que acepte existencias de su producto suele ser difícil, en especial en las temporadas en que hay mayor demanda, pero es preceptivo para alcanzar buenas cifras de ventas

La programación de juegos es una especialidad que suele comportar un margen de riesgo y gran competitividad

La programación por cuenta propia es un negocio arriesgado. En primer lugar, es probable que la retribución sea escasa, a menos que el juego sea un gran éxito. Los nombres más acreditados pueden recibir un anticipo y un porcentaje por los *royalties* correspondientes, pero es improbable que un programador desconocido reciba algo por adelantado y éstos, por lo general, son ínfimos. En segundo lugar, transcurrirá bastante tiempo antes de que se reciba algún pago. Preparar un juego para su lanzamiento puede llevar varios meses, de modo que cabe esperar que transcurra un año entre terminar el programa y recibir el primer pago. Y, por supuesto, no existe ninguna garantía de que le acepten el juego, aun teniendo un contrato. Todo contrato suele incluir una cláusula que afirma que el juego se debe completar a entera satisfacción de la casa de software, y la triste realidad es que algunos tipos de juegos pasan de moda con mucha rapidez.

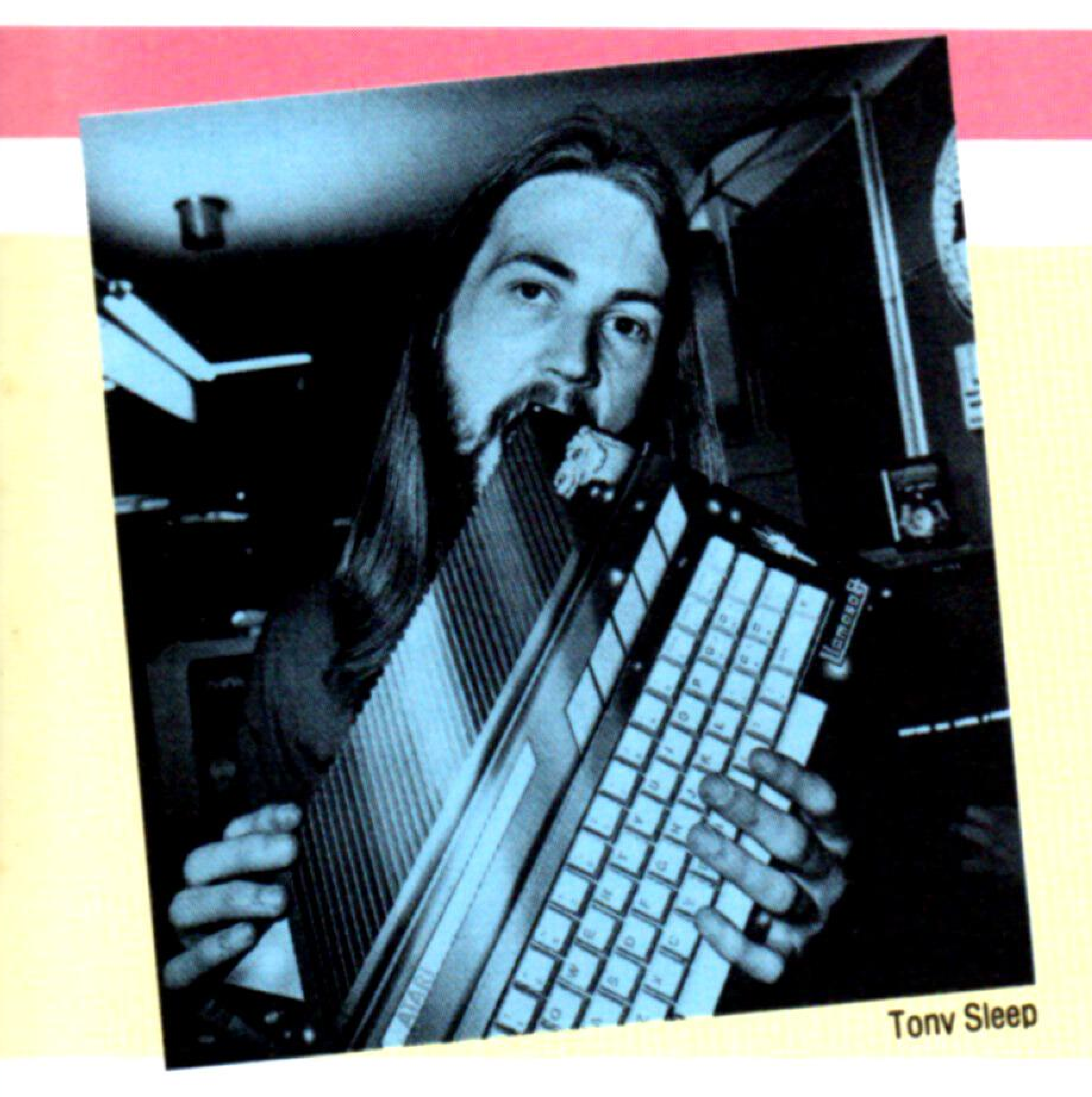
La mayoría de las casas de software de juegos operan fundamentalmente con programadores por cuenta propia, pero existen oportunidades de empleo. Si usted posee talento e ideas, la mayor parte de las casas de software aprovecharán la posibilidad de proporcionarle un puesto de trabajo. Las ventajas que ofrece el trabajo por cuenta ajena sobre el trabajo por cuenta propia son, en teoría, seguridad, posibilidades de promoción, formación y la sensación de formar parte de un equipo. En la práctica, la seguridad, las perspectivas de promoción y la formación distan mucho de estar garantizado.

A la vista de la regularidad con que aparecen y desaparecen las casas de software, es discutible que exista un empleo seguro en la programación de juegos. Obtener estadísticas fiables es difícil, porque muchas "casas de software" no son más que un nombre comercial que encubre una actividad desarrollada en un garaje o en un cobertizo de jardín. En Gran Bretaña, por ejemplo, hay probablemente entre 300 y 350 casas de software de juegos en existencia en cualquier momento dado. Cada año se crean alrededor de 200 o más, y en el mismo período desaparece aproximadamente la misma cantidad. Este proceso de equilibrio crea una falsa impresión de estabilidad: aunque la cantidad total de empresas permanece moderadamente constante, ¡la proporción de fracasos probablemente supere el 50 % del volumen de la industria!

La formación y las perspectivas de promoción dependen en gran medida de la empresa de que se trate. Un puñado de las compañías más importantes proporcionan adiestramiento formal, pero la inmensa mayoría no lo hace. La casa de software típica organiza a los programadores en equipos, con un programador senior como jefe de equipo. Los programadores junior pueden consultar al jefe de equipo para recabar su ayuda y su consejo. Una o dos empresas les pagan a los programadores junior para acudir a clases nocturnas en facultades locales, pero normalmente sólo abonan la matrícula del curso, de modo que en realidad el alumno lo cursa fuera del horario de trabajo.

En las empresas más pequeñas, las perspectivas de promoción son escasas o inexistentes. El director gerente lleva la empresa y los programadores escriben el software. Quizá haya un gerente de ventas/marketing para comercializar los juegos, pero normalmente el personal queda reducido a eso. En las empresas más grandes, la escalera de promoción habitual es de programador junior a jefe de equipo, y tal vez de allí a la gerencia si la persona posee aptitudes e inclinación para ello. La promoción depende casi exclusivamente de la capacidad; si el candidato es eficiente, será promocionado a jefe de equipo en un plazo de alrededor de seis meses. Si no evidencia aptitudes para ocupar el cargo, probablemente no le servirá de nada, por ejemplo, tener





Tony Sleep

cinco años de antigüedad en la empresa. Las cualificaciones, e incluso la experiencia, cuentan muy poco. Lo que los patrones desean ver es una prueba de su capacidad (es decir, un juego terminado), ideas renovadoras y conciencia del mercado. La mayoría de las casas de software afirman que prefieren cualificaciones formales, pero todas ellas coinciden en que la falta de las mismas no consituiría un obstáculo para un programador con talento. Y muchas prefieren un joven de 17 años con ideas a un hombre de 40 con experiencia.

El escritor de juegos, por supuesto, habrá de programar en lenguaje máquina. Por lo general se exige Z80 o 6502, y el 68000 está adquiriendo también importancia. El BASIC, lamentablemente, no lo llevará a ninguna parte. Asimismo, necesitará un conocimiento profundo de uno de los micros personales que tengan mayor aceptación en la actualidad, por ejemplo, Spectrum, Commodore 64, MSX o la gama Amstrad CPC.

Programador de juegos: hechos concretos

Cualificaciones exigidas
Las cualificaciones formales en informática son de ayuda, pero no suelen ser esenciales.

Lenguajes requeridos
Lenguaje máquina. Normalmente Z80, 6502 o 68000. Asimismo, se espera que demuestre un profundo conocimiento de los micros personales más populares.

Escala salarial
Sumamente variable. El sueldo de un no titulado suele estar de acuerdo con su capacidad y el tamaño de la empresa. Los titulados pueden esperar una remuneración equiparable a las tasas normales de la industria.

Formación proporcionada
La formación regular es muy rara. La mayoría de las empresas proporcionan un adiestramiento informal, a través del propio trabajo realizado bajo la supervisión de un programador senior.

Perspectivas de promoción
Ninguna en las firmas pequeñas. En las grandes, cabe esperar progresar a programador senior y posiblemente a la gerencia, a tenor de los méritos.

Vendiendo su juego

Supongamos que usted acaba de escribir un juego destinado a hacerse más famoso que *Pacman, Defender* o incluso *Space invaders*. ¿Qué hacer para venderlo? Esta lista de comprobación está diseñada para encaminarlo por la senda correcta.

Protéjase
La mayoría de las casas de software son de una escrupulosa honestidad. No obstante, esto no es así en el caso de unas pocas. Usted puede proteger su juego tomando una sencillas precauciones:

- Incluir una nota de *copyright* en la primera línea del código. De ser posible, valerse de los trucos de su máquina para asegurarse de que no se la pueda eliminar; de lo contrario, empotrar más notas de *copyright* a lo largo del juego.
- Incluir líneas redundantes, es decir, líneas que parezcan formar parte del juego pero que no tengan ningún efecto. Si el juego aparece sin su consentimiento, puede usar estas líneas redundantes como ayuda para establecer su *copyright*.
- Depositar una copia del programa en su banco, solicitando un recibo con la fecha.
- Conservar todo lo que escribió. Las primeras versiones del programa, los diagramas de flujo, trozos de rutinas, notas garabateadas.
- Llevar el programa a la casa de software y obtener un recibo firmado y fechado. Si utiliza el correo, enviarlo con acuse de recibo.

Acabarlo con todos los detalles
Las casas de software cuidarán con todo gusto de la presentación del programa. Quizá añadan su propia pantalla de títulos, escriban el folleto de instrucciones, añadan un cargador más veloz, etc. No obstante, esperan recibir el juego completo en todos los sentidos.

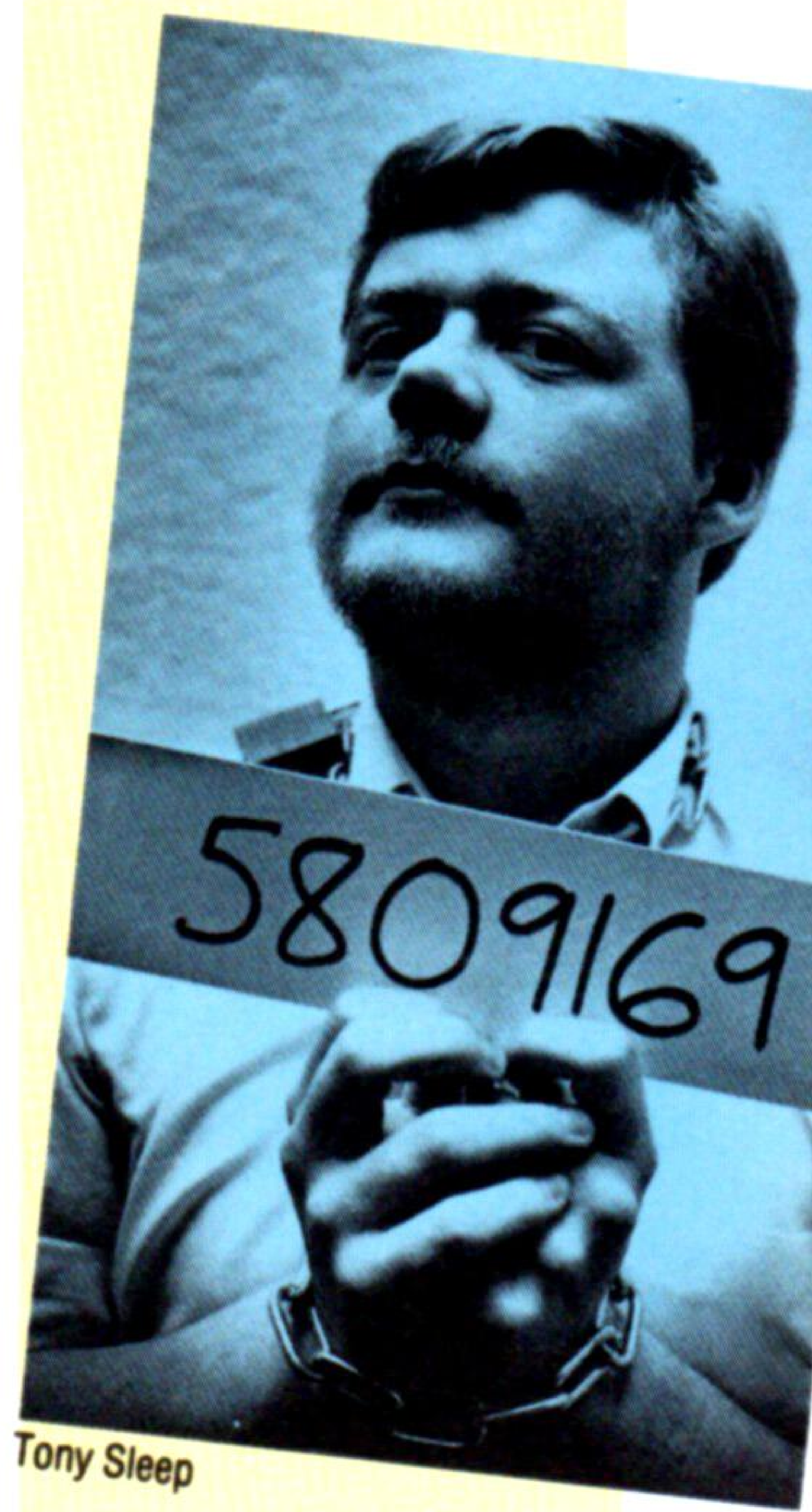
Preste atención a la presentación
Las casas de software reciben una gran cantidad de entregas no solicitadas. Todo cuanto pueda incluir para que la suya se destaque será de ayuda, como:

- Un resumen breve de 100 o 200 palabras.
- Instrucciones claras para cargarlo y jugar.
- Varias copias del juego por si se presentan dificultades de carga (disco antes que cinta).
- Un diagrama de flujo, si ayuda a esclarecer la estructura del juego.
- Detalles relativos a la máquina básica. Por ejemplo, 48 Kbytes de RAM, palanca de mando, unidad de disco, etc.

Marque todo con su nombre, dirección y número de teléfono en horas de trabajo.

No se comprometa de inmediato
Envíe su juego a tantas casas de software como sea posible, y no acepte la primera oferta en el instante en que se produzca. Si recibe una oferta, telefonee a algunas de las otras casas y hágales saber que le han hecho una oferta, pero sin ofrecer mayores detalles. Quizás ello los apure a evaluar su juego.

Nunca firme un contrato sin consejo profesional. En Gran Bretaña muchos abogados ofrecen consultas de tarifa fija, de alrededor de 20 minutos, por entre £5 y £10. Éste es un dinero bien invertido.



Tony Sleep

La experiencia cuenta
Dave Nicholls trabaja a jornada completa como programador para la Ram-Jam Corporation. El grueso de su trabajo consiste en convertir programas de un sistema a otro. Los antecedentes de Nicholls como programador de miniordenadores VAX y su experiencia comercial le facilitan la labor de pasar de un sistema a otro y de mantenerse al tanto de los nuevos desarrollos

Asignación de recursos

Finalmente analizaremos cómo interactúa el lenguaje con los sistemas operativos

Al c se lo utiliza principalmente, junto a otros fines, para escribir software de sistemas, sistemas operativos y utilidades, así como eficaz software de aplicaciones. Esto significa, entre otras cosas, que ha de existir una buena interface entre el lenguaje y el OS bajo el cual se está ejecutando, y que se han de poder programar funciones del mismo.

Una función particularmente útil cuando se escribe código que se ha de transferir entre sistemas diferentes en sizeof(objeto). Operador más que función, se le puede proporcionar tanto un nombre de variable como un nombre de tipo, y devuelve la cantidad de bytes utilizados para el almacenamiento de ese objeto.

Un ejemplo de su uso podría ser escribir código que funcione si int es de 16 o 32 bits. Con frecuencia se emplea junto con cuatro funciones (calloc(), malloc(), realloc() y free()), todas las cuales se ocupan de la asignación de memoria.

- **calloc ():**

```
char* calloc(número, tamaño)
int número,tamaño
```

La instrucción calloc() asigna, e inicializa en cero, una zona de memoria suficiente para retener un número de elementos, cada uno de ellos de la cantidad de bytes tamaño, devolviendo un puntero al primer elemento (o NULL si la memoria disponible es insuficiente). Por ejemplo, para reservar espacio para 50 enteros, podríamos usar:

```
p=calloc(50,tamañode(in));
```

- **malloc():**

```
char* malloc(bytes)
unsigned bytes;
```

Realiza una función similar, pero simplemente reserva la cantidad de bytes bytes sin inicializar. Con estas dos funciones, el valor devuelto es tan sólo un puntero a char, puesto que char siempre es un byte. Para utilizarlo realmente como un puntero a un objeto de un tipo diferente se lo debe *cast*.

- **realloc():**

```
char *realloc(p,bytes)
char *p;
int bytes;
```

El comando realloc() cambia el tamaño de la zona a la cual apunta p, a la cantidad de bytes bytes. Copiará el contenido del área antigua en el área nueva, pero puede o no reasignar una nueva sección de memoria, de modo que tras la misma los punteros de la zona ya no serán válidos.

- **free():**

```
free(p)
char *p;
```

Esta función libera un área de memoria que ha sido asignada por calloc, malloc o realloc para su reutilización. Podría ser un error desastroso utilizarla con cualquier otro valor de puntero.

Otra clase de funciones del c trata de la manipulación de errores. La mayor parte de los errores que pueden surgir en un programa en c se producen a raíz de la manipulación de entrada/salida o en las llamadas a la biblioteca. La biblioteca incluye una variable int estándar denominada errno y una lista de mensajes de error del sistema sys_errlist[], a la que se puede aludir directamente desde dentro del programa. Si se produce un error durante una llamada a la biblioteca, entonces se colocará un valor en errno para indicar esta circunstancia. Una llamada a:

```
perror(s)
char *s="mi mensaje de error";
```

visualizará en stderr tanto el mensaje dado como el de error del sistema correspondiente al valor actual de errno.

Cuando ocurren errores en entrada/salida en algunas ocasiones producirán un error del sistema normal, pero no siempre se puede confiar en ello. La función int ferror(puntero_archivo) comprobará si se ha producido un error al leer o al escribir en el archivo dado, devolviendo cero si se ha producido un error y no cero en caso contrario. La condición de error puede impedir el acceso al archivo, de modo que se proporciona la función clearerr (puntero_archivo) para restaurar la condición de error.

También se proporcionan varias funciones para la administración general de archivos en disco y directorios. Los detalles relativos a las mismas se deben incluir en su compilador, pero casi con toda seguridad incluirán access(), que comprueba la modalidad de acceso de un archivo o directorio mencionados; chmod(), que cambia la modalidad de acceso; y chdir() para cambiar el directorio de trabajo actual.

El programa de ejemplo es relativamente largo e implica casi todas las facilidades del c que hemos visto hasta ahora, incluyendo algunas (como llamadas recursivas a funciones) de las que no hemos hablado explícitamente, así como malloc, para obtener espacio de almacenamiento para mantener una lista enlazada. El programa tiene por objeto crear un índice para un archivo de texto grande; en otras palabras, una lista de todas las palabras utilizadas y los números de página en las que aparecen.

Las palabras se conservan en una lista enlazada que crece a medida que se va encontrando cada palabra. Cada palabra lleva asociado un puntero a otra lista enlazada de números de página para cada palabra.

El c es un lenguaje muy interesante de emplear y de amplia difusión. Es uno de los pocos lenguajes que ofrece la posibilidad de escribir un programa en un micro pequeño que también funcione en micros grandes, en minis e incluso en ordenadores centrales. Si usted no ha intentado aún utilizar el c, decididamente bien vale el esfuerzo: sus cualidades son muchas y su difusión es mayor cada vez.



Compilador de índice

```
/*En archivo index.h*/
#define NULL 0
# define TAMAÑOMAXIMOPALABRA 20
typedef char ENTRY[TAMAÑOMAXIMOPALABRA]:
/*Cada elemento del índice estará compuesto
por la entrada real, un puntero a la lista de
números de página y un puntero al siguiente
elemento*/
struct número__página
{
   int pn;
   struct número__página*psiguiente;
}
typedef struct número__página página;
typedef página*penlace;
struct elemento__índice
{
   ENTRY entrada;
   penlace páginas;
   struct elemento__índice * salir
}
typedef struct elemento__índice elemento;
typedef elemento * enlace;
/* ahora podemos aludir a un artículo de la
lista como un "elemento" y un puntero a un
elemento se denomina un "enlace"*/
/* en archivo index.c*/
# include stdio.h
# include string.h
# include index.h
# define LPP=66;/* líneas por página */
main(argc,argv)
int argc;
char*argv[];
enlace head;
{
   FILE enarchivo;
   int lc=0,pc=1,enpalabra=0;
   ENTRY siguientepalabra;
   char*nw;
/* inicializar la lista con una primera entrada
ficticia (facilita las cosas!)*/
   head=nueva__entrada ("",NULL,0);
/* poner nombre archivo a índice */
   if(argc!=2)
   {
      fprinft(stderr,"\nusage es %s
nombrearchivo\n", *argv);
      salir(l);
   }
   if (enarchivo=
      fopen(*++argv,"r")==NULL)
      fprintf(stderr,"n\archivo no hallado
%s\n",
      * argav);
      salir(l);
}
   nw=siguientepalabra;
   while(c=getc(enarchivo)!=EOF)
   {
      if(c=='\n')
/* sumar uno al contador de líneas y
comprobar si final de página */
   {
      lc+=1;
      if(lc>LPP)
      {
         pc+=1;
         lc=0;
      }
   }
   else if (enpalabra)
   {
      if(isalpha(c))
         *nw++=c;
      else
      {
         enpalabra=0;
         *nw='\o';
         insert(siguientepalabra,head,pc);
         nw=siguientepalabra;
      }
   }
   else
   {
      if(isalpha(c))
      {
         enpalabra=1;
         *nw++=c;
      }
   }
}
if(enpalabra)
{
   *nw='\o';
   insert(siguientepalabra,head,
   pc);
}
   visualizar__índice;
}
insert(e)
ENTRY e;
{
insert(e,l,pnum)
ENTRY e;
enlace l;
int pnum;
static enlace últimol;
{
   últimol=head;
   if(l==NULL)
   {
      últimol—>siguiente=siguiente__
entrada(e,l,pnum);
      return;
   }
   else
   {
      int s;
      s=strcmp(e,l—>entrada);
      if(s==0)
/* palabra ya presente de modo que añadir
número página */
      {
         añadir__número__página(pnum,
         l—> páginas);
         return;
      }
      else if(s>0)
/* se ha ido demasiado lejos de modo que
insertar nuevo nudo después del último de la
lista*/
      {
         últimol—> siguiente=nueva__
entrada(e,l,pnum);
         return;
      }
      else
      {
/* no hallado aún de modo que recorrer la lista
utilizando una llamada recursiva a insert*/
         últimol=l;
         insert(e,l—> siguiente,pnum);
         return;
      }
   }
}
enlace nueva__entrada(e,l,pnum)
ENTRY e;
enlace l;
int pnum;
{
/* tomar espacio suficiente para nueva entrada
   utilizando malloc */
   enlace nueval;
   nueval=(enlace)malloc(tamañode(elemento));
/* observe cast para convertir puntero char
   devuelto por malloc */
   nueval—>entrada=e;
   nueval—>siguiente=l;
   nueval—>páginas= (penlace)malloc(tamañode(página));
   nueval—>páginas—>pn=pnum;
   nueval—>páginas—>psiguiente=NULL;
   return(nueval);
}
añadir__número__página(pnum,pl)
int pnum;
plink pl;
{
/* hallar final de lista números de página */
   while (pl—> pnext)!=NULL)
      pl=pl—> pnext;
   pl—> psiguiente=(plink)malloc(tamañode(página));
   pl—> psiguiente—> psiguiente=NULL;
   pl—> psiguiente—> pn=pnum;
   return;
}
visualizar__índice;
enlacel;
penlace pl;
{
   l=head—> siguiente;
   while(l!=NULL)
   {
      printf("%s\t",l—> entrada);
      pl=l—> páginas;
      while(pl—> siguiente!=NULL)
      {
         printf("%d4,",pl—> pn);
         pl=pl—> siguiente;
      }
      printf("%d\n",pl—> pn);
   }
   return;
}
```

Material fuente

Consideremos cuáles son las características que debe reunir un buen ensamblador y examinemos tres conocidos paquetes para ordenadores personales

Si bien un ensamblador es esencial para el programador serio, es sólo una de las muchas herramientas que se requieren. El ensamblador convierte el *código fuente* (escrito con los mnemotécnicos y símbolos del lenguaje assembly) original de la máquina a *código objeto* (el auténtico lenguaje máquina). También es necesario un programa monitor o depurador para verificar la corrección del programa y localizar sus fallos.

La mayoría de los ensambladores siguen el lenguaje assembly definido por el fabricante de la CPU, pero los ensambladores pequeños o sencillos a menudo utilizan una sintaxis no estándar para simplificar sus tareas. Para cargar un valor inmediato en el registro A de un 6502, por ejemplo, el lenguaje assembly correcto es LDA # $20, mientras que un ensamblador sencillo podría utilizar LDAIM $20. No obstante, se deben evitar los ensambladores no estándares, porque pueden ser difíciles de aprender y habrá de convertir los archivos fuente existentes antes de que se ensamblen.

Los ensambladores más sencillos están incorporados en programas depuradores. Denominados *ensambladores en línea*, permiten entrar y ensam-

Hisoft Devpac

Para el Amstrad CPC 464, 664 y 6128, Spectrum 48 K, micros CP/M y MSX



El conocido Devpac de Hisoft es una herramienta de desarrollo completa, con ensamblador, editor y depurador de código máquina. Los programas se desarrollan utilizando líneas numeradas similares al BASIC, pero también posee facilidades completas de edición: renumerar, suprimir y desplazar líneas, hallar y reemplazar, etc. El Devpac es del tipo de dos pasos, con una amplia gama de directivas y exhaustivas capacidades aritméticas. Una amplia gama de opciones de ensamblaje proporcionan al programador un control flexible del modo en que se produce el código. Tanto el programa ensamblador/editor como el depurador son reubicables, lo que permite cargarlos en la memoria en un punto adecuado para el programa que esté desarrollando. Si se cargan ambos programas al mismo tiempo, se enlazarán, permitiéndole a usted saltar fácilmente desde el ensamblador al depurador cuando ello sea necesario. El depurador imita el "panel frontal" de los ordenadores centrales produciendo una visualización de pantalla casi completa, mostrando el contenido de un bloque de memoria, los registros de la CPU y la próxima instrucción a ejecutarse. Además de las facilidades habituales, Hisoft proporciona facilidades de puntos de parada, ejecución paso a paso y seguimiento, convirtiéndolo en una herramienta ideal para depurar programas. Su completa visualización en pantalla es muy adecuada para los principiantes. Devpac viene con un detallado manual de 32 páginas. Aunque ni éste ni las instrucciones Devpac propiamente dichas son todo lo claras que deberían ser, las omisiones de información relativa a todo cuanto debe saber el programador son escasas. El Devpac es un paquete exhaustivo y bien producido

Tipo de ensamblador

Dos pasos, mnemotécnicos estándares

Límites

Código fuente: Limitado por la RAM, pero hay disponible enlace de archivos
Tamaño de la tabla de símbolos: especificado por el usuario

Directivas de ensamblador

ORG	Establecer el origen del programa
EQU	Asignar un valor a un símbolo
DEFB	Almacenar valor(es) de 8 bits
DEFW	Almacenar valor(es) de 16 bits
DEFS	Reservar espacio de almacenamiento
DEFM	Almacenar serie ASCII
ENT	Marca la dirección de comienzo del programa para la instrucción RUN del ensamblador
END	Marca el final del programa fuente
IF...ELSE	Ensamblaje condicionado
*E	Salto de página en impresora
*H	Define encabezamiento en listado
*S	Hacer una pausa en listado
*L	Activar y desactivar listado
*D	Dar direcciones de listado en decimal
*F	Incluir archivo fuente preescrito en este punto
*T	Escribir código objeto en cinta en vez de en la memoria

Opciones de ensamblaje

Producir tabla de símbolos o no
Generar código objeto o no
Producir listado assembly o no
Colocar código objeto tras tabla de símbolos o en dirección ORG
Comprobar que el código objeto no se sobreescriba en Devpac o no comprobarlo

Aritmética

+, −, *, /, MOD, AND, OR, NOT y XOR. Decimal, hexa y binario

blar directamente en la memoria una línea de lenguaje assembly. Pero no permiten que el programador asigne etiquetas y símbolos a las direcciones y valores utilizados en el programa, lo que los hace adecuados sólo para realizar cortas correcciones en programas que ya se hallen en la memoria.

Con frecuencia los depuradores incluyen facilidades tales como inserción de puntos de parada, ejecución paso a paso y seguimiento de la ejecución de un programa. La inserción de un punto de parada en el programa a depurar genera en él una instrucción de salto que dará el control al depurador cuando sea ejecutada y se suprimirá ella misma después de su ejecución. De este modo, se puede comprobar en cualquier punto el estado de los registros y las posiciones de memoria.

Los ensambladores completos pueden ser, como se ha dado en llamarlos, de *un solo paso* o de *dos pasos*. El ensamblador de un solo paso más sencillo lee el código fuente sólo una vez, de modo que si el programa salta hacia adelante hasta una instrucción etiquetada, el ensamblador no sabrá a qué dirección se refiere la etiqueta. En este caso, el ensamblador retrocederá y colocará la dirección una vez que la haya descubierto; sin embargo, la dirección real no aparecerá en el listado assembly. Es más común el uso de un ensamblador de dos pasos; éste lee el programa fuente una vez, comprobando su sintaxis y construyendo un listado completo de los símbolos utilizados en él antes de volver a leerlo y producir un listado assembly ya completo.

Todos los ensambladores poseen su propio juego de *directivas* o *pseudo-ops*. Son instrucciones que no forman parte del juego de instrucciones de la CPU y que no se traducen a código máquina. En cambio, indican al ensamblador que debe llevar a cabo alguna tarea específica, como conectar la impresora o reservar un bloque de espacio de memoria dentro del código para almacenamiento de datos. Las directivas varían de un paquete a otro, tanto en lo que hacen como en sus nombres y sintaxis. La mayoría de las directivas están bastante estandarizadas, pero las inusuales pueden representar un problema si está intentando lograr que un programa escrito para un ensamblador trabaje con otro.

La primera directiva con la cual se encontrará es ORG, que indica al ensamblador dónde se empezará a cargar en la memoria el programa a ensamblar. Asimismo, todos los ensambladores poseen una forma de definir símbolos: COUNT EQU 5, establecería el símbolo COUNTen 5. Además, es muy útil un buen juego de *directivas de datos*, que son instrucciones que le permiten establecer las áreas de almacenamiento de sus programas. Por lo general hay instrucciones para almacenar valores determinados (DEFB, DEFW) o series (DEFM), y para dejar libres algunos bytes para el programa (DEFS).

Operaciones aritméticas

La mayoría de los ensambladores pueden realizar aritmética, y es útil que proporcionen una gama completa de operadores: +, −, /, *, AND, OR, NOT y MOD. Este último, MOD (o un equivalente), es virtualmente indispensable, puesto que con frecuencia los programadores necesitan el ensamblador para dividir un valor de 16 bits en dos valores de ocho bits. Los buenos ensambladores también son capaces de manipular números en decimal, hexa y binario, así como de evaluar un carácter ASCII (de

Picturesque Editor/Assembler

Para el Spectrum de 16 K y 48 K

El Editor/Assembler de Picturesque viene siendo desde hace ya tiempo el favorito de los programadores del Spectrum, profesionales o aficionados indistintamente. Constituye un ensamblador y editor Z80 de dos pasos en el Spectrum, y opcionalmente se lo puede utilizar con el paquete Monitor de Picturesque. Opera tanto en máquinas de 16 K como de 48 K, si bien en la versión de 16 K sus facilidades son más restringidas. Para atenuar el problema que supone mostrar listados anchos en assembly en la visualización de 32 columnas del Spectrum, el programa simula una pantalla de 40 columnas.

La versión para 48 K posee la muy importante capacidad de ensamblar desde archivos de código fuente almacenados en cinta o en microdrive. En efecto, ello dispone de lugar para programas fuente de hasta 95 Kbytes de longitud y, por consiguiente, la creación de programas objeto grandes, como los necesarios para juegos comerciales. Ésta es una facilidad estándar en los ensambladores para micros de gestión. Muchos ensambladores para ordenadores personales se limitan a programas suficientemente cortos para que tanto su código fuente como su código objeto se hallen en la memoria al mismo tiempo. Picturesque proporciona un excelente manual de 56 páginas para el Editor/Assembler, que contiene información clara y detallada sobre todos los aspectos concernientes al paquete. A pesar de que existen alternativas más recientes que ofrecen más facilidades de ensamblaje, Picturesque es un excelente patrón y un paquete muy capaz por derecho propio

Tipo de ensamblador

Dos pasos, mnemotécnicos estándares

Límites

Código fuente: hasta 95 K utilizando microdrive
Tamaño de la tabla de símbolos: limitado por la memoria libre

Directivas de ensamblador

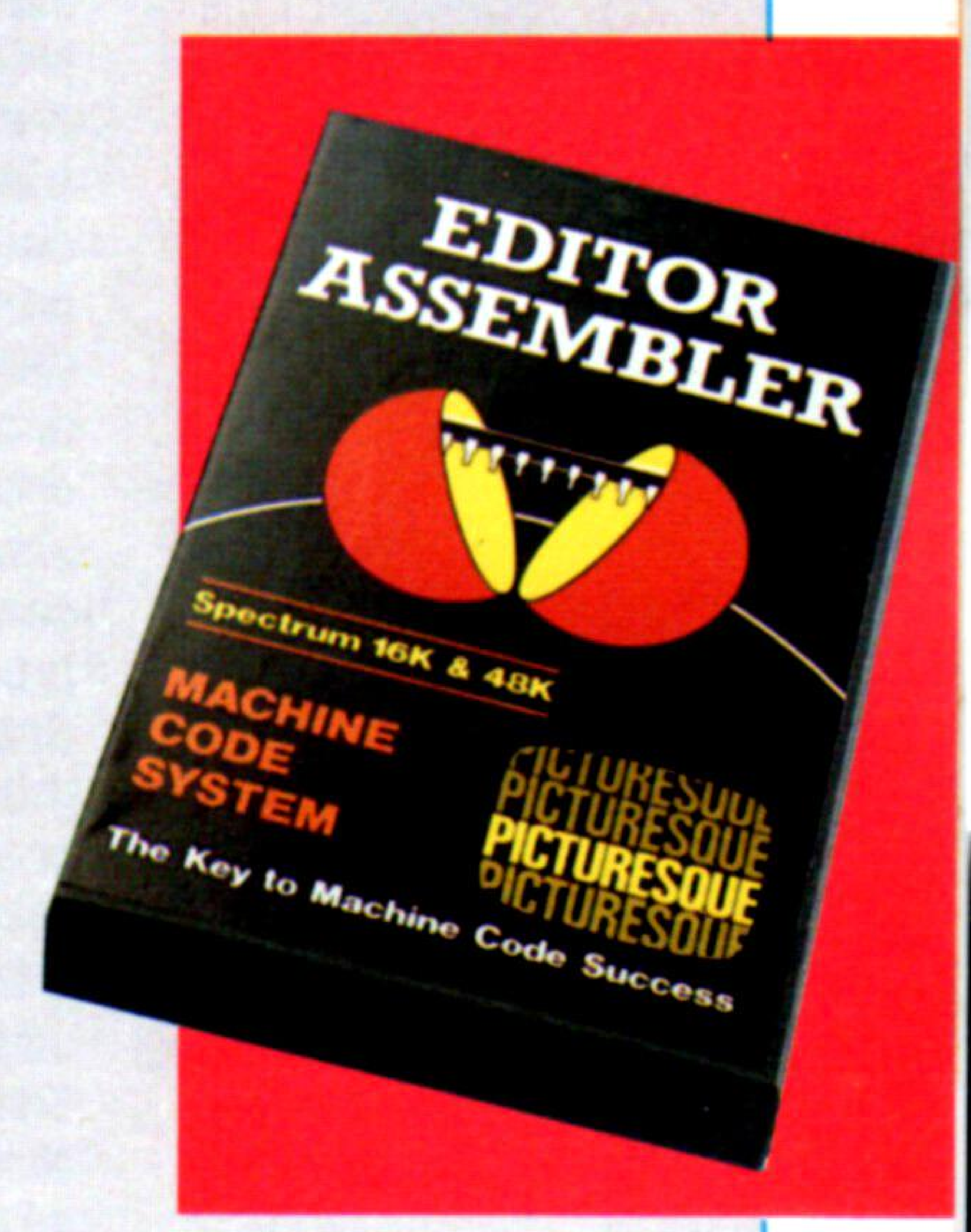
ORG	Establecer origen del programa
EQU,	Asignar valor a símbolo
DEFL	Como EQU, pero el símbolo se puede reasignar varias veces
DEFB	Almacenar valor(es) de 8 bits
DEFW	Almacenar valor(es) de 16 bits
DEFS	Reservar área de almacenamiento
DEFM	Almacenar serie ASCII
PRNT	Conectar y desconectar impresora
END	Marca el final del programa fuente

Opciones de ensamblaje

Ensamblaje desde memoria, cinta o microdrive
Listado assembly en impresora o en pantalla

Aritmética

+, −, < y > (bytes *high* y *low* de valores de 16 bits).
Hexa y decimal



modo que, por ejemplo, usted podría utilizar el símbolo A en lugar de tener que consultar por sí mismo su valor en una tabla ASCII).

Cuando usted solicita que se ensamble un programa, por lo general se le ofrece la posibilidad de especificar algunas opciones en cuanto al modo de llevar a cabo el ensamblaje. Éstas pueden incluir que se produzca o no un listado y tabla de símbolos, y que ésta sea visualizada o no en la pantalla, sea enviada a la impresora o se escriba en un archivo en disco o cinta. Una facilidad muy útil es un ensamblaje experimental, en virtud del cual el programa se ensambla pero no se carga en la memoria ningún código máquina (¡ésta es una forma rápida de comprobar que el programa se ensamble pero sin ensamblarlo realmente!).

Los mejores paquetes ensambladores permiten el ensamblaje tanto condicional como el uso de macros. Las directivas condicionales permiten ensamblar secciones de listado sólo si determinadas condiciones son verdaderas. Esto significa que un listado fuente puede generar varias versiones del código objeto según las exigencias. Una macro permite definir un nuevo mnemónico a partir de algunos de los ya existentes. Cuando se ensambla la macro, el ensamblador sustituye automáticamente las instrucciones correctas. Mediante la construcción de una biblioteca de macros para efectuar tareas determinadas (como imprimir mensajes, abrir archivos de datos, etc.), se pueden escribir programas con suma rapidez. No obstante, no son tan eficaces como los programas codificados a mano en cuanto a velocidad de ejecución y tamaño.

Aparte de las importantes directivas, la otra característica fundamental de los ensambladores es cómo están organizados. Muchos almacenan en la memoria tanto el código fuente como el código objeto, lo que limita el tamaño máximo del programa que se puede escribir. Una solución implica poder ensamblar a partir de un programa fuente en almacenamiento, lo que por lo general es práctico sólo en máquinas con discos o cintas rápidas, tales como la Sinclair Microdrive. Pero ésta es la única forma de crear programas grandes.

Además de las especificaciones del ensamblador, se debe comprobar su grado de rapidez. Un programa se ensamblará muchas veces a lo largo de su desarrollo, de modo que con frecuencia es preferible un pequeño ensamblador rápido a uno más lento con muchas facilidades. También es importante la calidad del editor suministrado. Lo ideal es un editor de pantalla completa similar a un procesador de textos, si bien muchos programas se basan en editores de línea.

Watford ROMAS

BBC Modelo B

ROMAS es un "ensamblador cruzado" profesional para el BBC B. Está diseñado para permitir el desarrollo de programas no sólo para el BBC sino para una amplia gama de micros. Se utiliza la directiva PROC para indicar al ensamblador para qué CPU está ensamblando. Actualmente el ROMAS soporta procesadores 6502, 6511, 8085, Z80, Z8, 6809, 65C02, 8041 y 8048, por lo cual es adecuado para casi todos los ordenadores de ocho bits y dispositivos controlados por microprocesador. ROMAS incluye utilidades que permiten la transmisión de programas objeto a otros micros y programadores EPROM. Además de ser un ensamblador cruzado, el ROMAS soporta ensamblaje condicional y posee una gama de directivas diseñadas para ayudar a mantener grandes programas y producir versiones de los mismos para ejecutar en diferentes máquinas. Por ejemplo, las directivas TYPE e INPUT permiten que el ensamblador solicite al usuario valores para símbolos durante el ensamblaje. En unión con las condicionales, es posible que al ensamblar un programa se pregunte "¿qué máquina? 1 = Spectrum, 2 = Amstrad, 3 = MSX" y luego se produzca la versión correcta del programa. ROMAS ensambla desde un archivo en disco y vuelve a escribir el código objeto otra vez en archivo en disco, que posteriormente será comprobado en el BBC, un segundo procesador BBC o el micro para el cual fue diseñado. El paquete incluye un editor en pantalla denominado EDT, que soporta macroinstrucciones; se pueden construir operaciones complejas a partir de instrucciones ya existentes. Indudablemente, ROMAS es uno de los editores más sofisticados que existen en la actualidad para un micro pequeño, incluyendo un manual de 156 páginas y un programa fuente de muestra (un desensamblador BBC). Si bien la mayor parte de los usuarios se sentirán satisfechos con el ensamblador incorporado en el BBC BASIC con la adición de un programa monitor, el ROMAS es una alternativa profesional para quienes desarrollan software seriamente



Liz Heaney

Tipo de ensamblaje

Ensamblador cruzado de dos pasos, mnemotécnicos estándares

Límites

Código fuente: limitado por la capacidad del disco
Tamaño de la tabla de símbolos: 7 K, pero se puede ampliar

Directivas de ensamblaje

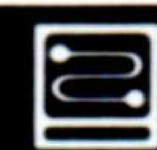
ORG	Establecer origen programa
EQU	Asignar valor a símbolo
DB,DW,DS	Almacenar valores de 8 y 16 bits, series ASCII y reservar espacio
IF,ELSE,ENDIF	Ensamblaje condicional
END	Marca el final de un programa
EXTEND	Seguir ensamblando desde archivo fuente mencionado
HIMEM,LOMEN,MODE	Como en el BBC BASIC
TYPE,INPUT,GET	Interactuar con el usuario durante el ensamblaje
TITLE,PAGE,EJECT, LIST,TAB,WIDTH	Controles de listado
PROC	Seleccionar procesador para el cual ensamblar

Opciones de ensamblaje

Producir listado assembly en pantalla o en impresora

Aritmética

*, /, REM(modulo), +, −, AND, OR, NOT, XOR, desplaz. izq., desplaz. der. Decimal, hexa, octal y binario



Breve pausa

En el capítulo anterior estudiamos los mecanismos básicos de E/S en serie. Esta vez analizaremos las E/S a través de interfaces paralelas

Comenzaremos con una mirada a un ejemplo de salida en serie con una subrutina llamada mensaje. Esta rutina tiene la dirección del mensaje a ella pasado mediante el registro de dirección A3. Así, por ejemplo:

```
LEA TEXT,A3        establece un puntero hacia el
                   mensaje TEX
JSR MENSAJE        salida del mensaje
```

dará salida al mensaje TEXT almacenado así:

```
TEXT:DC.B          'Mi Computer',$00
```

donde DC.B es una directiva del ensamblador para declarar el espacio de memoria que albergará el texto 'Mi Computer'. El byte nulo $00 representa el fin del mensaje, e informa a la subrutina mensaje que ya no hay más texto.

La subrutina para dar salida a todos los bytes en la tabla TEXT usará la subrutina OUTCH, que ya conocemos de un capítulo anterior. En el ejemplo, OUTCH empleaba el bit "preparado" para dar salida a los caracteres uno a uno tan pronto como estuviera lista la ACIA para enviar el carácter siguiente. La subrutina mensaje tendrá, pues, el siguiente aspecto:

```
MENSAJE: MOVE.B  (A3)+D0   toma el siguiente
                           byte de mensaje
         BEQ     DONE      comprueba el final
                           del mensaje
         JSR     OUTCH     salida de
                           caracteres
         BRA     MENSAJE   bucle hasta llegar
                           al final del
                           mensaje
   DONE: RTS
```

Cada byte del mensaje se copia en D0 (el registro de datos empleado como parámetro de valor para OUTCH), empleando A3 como puntero con direccionamiento indirecto posincremental. Si el byte es cero, se abandona la rutina mediante la etiqueta DONE; de lo contrario se da salida a un carácter a través de OUTCH.

Si precisamos dar salida a datos en el modo paralelo, en el que todos los bits de los datos aparecerán simultáneamente, el chip PIA nos proporciona las facilidades. Sin embargo, dado que éste es un chip interface para uso general, debemos configurarlo en la forma adecuada a la configuración particular de nuestro hardware. Esto se parece al chip ACIA, donde debemos establecer la velocidad de transmisión y los formatos de los bytes.

Para el PIA, habremos de configurar las ocho líneas paralelas de *ambas* mitades del chip (A y B) para que sean o bien líneas de salida o bien de entrada (es decir, debemos configurar la dirección de los datos). Para ello escribiremos en el bit 2 del registro de control (CRA o CRB) para indicar que deseamos establecer la dirección de los datos en un registro de dirección de datos (DDRA o DDRB). Por ejemplo:

```
CONFIGPIA: CLR.B   PIACRB        pone a cero
                                 el bit de
                                 registro de
                                 control
           MOVE.B  #$FF,PIADDRB  establece la
                                 dirección de
                                 los datos en
                                 salida
           BSET    #2,PIACRB     retorna al
                                 registro
                                 normal de
                                 datos
           CLR.B   PIACRA        da para A
                                 la dirección
                                 del sentido
                                 de los datos
           CLR.B   PIADDRA       establece el
                                 sentido de los
                                 datos en
                                 salida
           BSET    #2,PIACRA     vuelve el
                                 registro
                                 normal de
                                 datos
```

Este fragmento establecerá todo el lado A para entrada, y el lado B para salida. La transmisión de los datos a los dispositivos paralelos se ejecutará con:

```
MOVE.B    D0,PIADRA    salida del contenido de D0
```

y para la entrada tendremos lo siguiente:

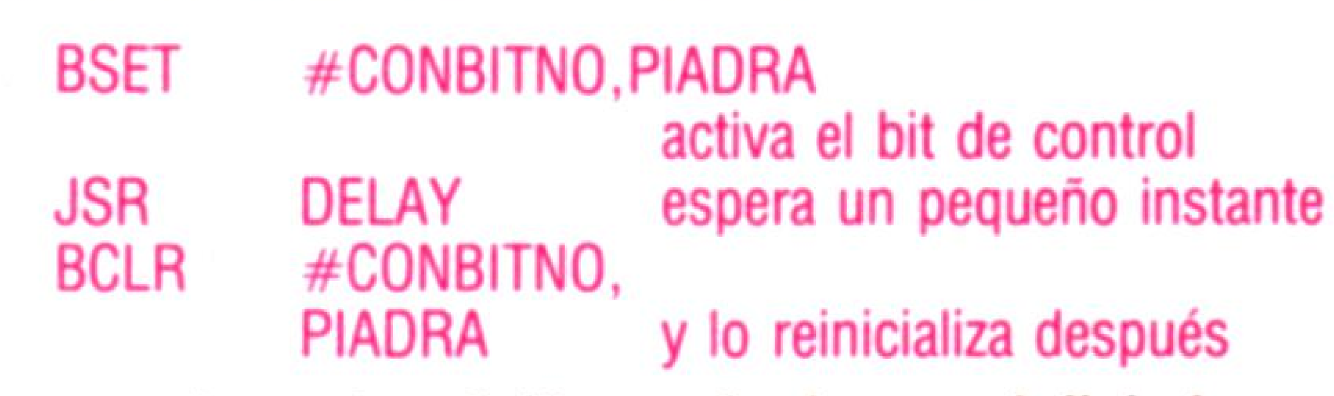
```
MOVE.B    PIADRA,D1    lee la entrada en D1
```

Obsérvese que todas las direcciones PIA a las que nos referimos deben establecerse inicialmente, como por ejemplo:

```
PIADRA    EQU    $30051
PIACRA    EQU    $30053
```

Obsérvese también que el significado de los datos leídos o escritos hacia el PIA dependen del tipo de dispositivo que se conecta eléctricamente a los canales digitales del chip (p. ej., podríamos tener una visualización de siete segmentos conectada, como se muestra en el diagrama donde puede verse que todo dígito decimal puede construirse mediante el segmento adecuado). Para visualizar el número 3, pongamos por caso, deberemos encender los segmentos 1, 4, 2, 5 y 3.

No sólo necesitaremos transmitir datos al dispositivo periférico sino que se puede necesitar el control de su función eléctrica. En el caso propuesto de una visualización con siete segmentos, puede que necesitemos fijar los datos en el dispositivo mediante algunos de los bits de reserva escritos en modo salida dentro de la palabra de datos. A menudo estas señales de control simulan el pulso de un "reloj" eléctrico, pudiendo traducirse en una activación y desactivación del bit de control. Así:



```
BSET    #CONBITNO,PIADRA
                        activa el bit de control
JSR     DELAY           espera un pequeño instante
BCLR    #CONBITNO,
        PIADRA          y lo reinicializa después
```

proporciona el "reloj" a cualquier canal digital que se asigne a CONBITNO en la dirección PIADRA.

Movimiento paralelo

El dibujo inferior muestra la estructura de un típico chip PIA, utilizado para proporcionar facilidades de entrada/salida al 68000. El chip proporciona dos puertas, asignadas convencionalmente: una para la entrada y otra para la salida. Cada puerta tiene sus propios registros de control, datos y dirección de datos. Una aplicación típica puede ser manejar una visualización de siete segmentos LED como el que se muestra en el dibujo (izquierda)

Caroline Clayton

Interrupciones

Aunque el tema de las interrupciones no siempre es bien entendido, la finalidad del uso de interrupciones en un sistema de ordenador tiene que ver con el uso eficiente de la CPU y el poder responder a eventos externos. Por ejemplo, puede que no deseemos que la CPU pierda el tiempo mientras se imprimen los caracteres, como en el ejemplo de la subrutina OUTCH. También necesitamos, para poder hacer otra cosa, saber cuándo se ha terminado de imprimir un carácter para enviar el siguiente disponible.

La situación es aún peor cuando el programa está a la espera de una entrada; digamos, por ejemplo, que del teclado. La eficiencia del sistema depende, como es obvio, de la velocidad de escritura y también de otras tareas asumidas por la CPU, tales como dar salida hacia la impresora en paralelo al tiempo que espera una entrada del teclado.

Para obtener esta operación en paralelo, debemos organizar la secuencia lógica de eventos dentro de la máquina para asegurarnos de que no perdemos el control del programa o algún dato. Veamos lo que se necesita.

- *Guardar el estado del ordenador.* Necesitamos hacer esto para poder volver al programa que hemos interrumpido sin ningún efecto notable o pérdida de datos (lo que sí perderemos inevitablemente es tiempo). Pero primero definiremos qué es lo que constituye el "estado" del ordenador.

Podemos considerarlo como el área total para programas y datos del programa del usuario con su estado de registro y contador del programa. En la práctica, y puesto que en general no se esperarán cambios en el programa del usuario desde la fuente de interrupciones, basta con guardar tan sólo el contador del programa (PC) y el registro de estado (SR). Esto nos dará una idea cabal del estado del programa interrumpido.

- *Localizar la fuente de interrupción.* Esto es necesario porque probablemente tendremos conectados al ordenador más de un dispositivo y necesitamos saber qué rutina de "servicio" se ha de ejecutar.

- *Inhibir interrupciones de otras fuentes.* Necesitamos hacerlo porque si llega otra interrupción mientras atendemos a una ya producida se pueden perder datos. Esto, sin embargo, sólo puede suceder si el tiempo invertido en atender la segunda interrupción ha sido mayor que el tiempo entre la llegada de los datos en la primera interrupción.



• *Entrar la rutina de servicio de la interrupción.* Necesitamos un mecanismo que haga esto, ya sea automáticamente o ejecutando una instrucción específica.

• *Volver al programa interrumpido.* Finalmente, después de haber atendido la interrupción, deberemos volver al estado exacto del programa inmediatamente anterior a la interrupción.

Veamos cómo se consigue esta organización lógica en el 68000. Ante todo, el estado de la máquina se guarda mediante la pila donde se almacena primero el PC del usuario (como palabra larga completa) y después el SR. Con este mecanismo, las interrupciones pueden anidarse de tal modo que se pueda interrumpir una rutina de tratamiento de interrupción.

Si la rutina de tratamiento de interrupción fuera a emplear algunos de los registros que ya hemos señalado para un propósito específico, la rutina puede salvar estos registros a la entrada y restaurarlos a la salida. La instrucción MOVEM hará esto en el 68000 con la mayor sencillez para nosotros:

```
MOVEM   D1,D3,-(SP)   apila los registros
                      de datos
                      código de servicio
                      de interrupción
MOVEM   (SP)+,D1,D3   y los restaura
```

El problema de localizar la fuente de interrupción es también fácil en el 68000, dado que a cada interrupción se le asigna generalmente una sola posición de memoria llamada *vector*, que puede considerarse que contiene un puntero hacia la dirección de la rutina de servicio. Naturalmente, pueden existir varios dispositivos en un determinado vector, en cuyo caso es necesario un *pooling* para establecer cuál es el dispositivo interruptor. Pero éste no suele ser el caso del 68000, por lo general.

La necesidad de arbitrar entre dispositivos interruptores también es atendida por usted. Y esto porque el 68000 asume la prioridad hardware de la fuente interruptora, es decir, *sólo* obtendrán la atención de la CPU las interrupciones de mayor prioridad. Esto significa que las fuentes de datos de alta velocidad tendrán una relativa alta prioridad en un sistema de multiinterrupciones, de modo que no se pierda dato alguno.

Una vez que la interrupción ha atraído la atención de la CPU, se carga en el PC el contenido del vector de interrupción, y se entra en la rutina de servicio de interrupciones. El dispositivo es atendido cargando los datos de entrada en un registro y de allí a algún buffer, o evacuando los datos de un buffer llevándolos a un dispositivo de salida.

La única acción que resta por hacer ahora es volver de la rutina de servicio al programa interrumpido. Esto se consigue casi de idéntica manera que una vuelta de subrutina, empleando esta vez una instrucción RTE y no una RTS. La instrucción RTE vuelve a cargar el PC y el SR automáticamente desde la pila de sistema, por lo que todas las rutinas de servicio deben concluir con esta instrucción. Significa también que en ese momento A7 (el puntero de la pila de sistema) debe estar apuntando a los registros del programa interrumpido. Entonces si usted coloca datos en la pila de sistema, ¡asegúrese de que son sacados de allí antes de ejecutar la instrucción RTE! O quizá más apropiadamente debería usar otra pila, dado que con el 68000 es posible emplear como pila cualquiera de los registros de direcciones.

Rutina de servicio

Veamos ahora una rutina de servicio de interrupciones típica donde tendremos también que hacer *pooling* de los dispositivos para encontrar la fuente interruptora. Dentro de nuestro sistema teórico, tenemos dos posibles fuentes de interrupción en lo que se llama nivel 4 (una línea de interrupción hardware de prioridad 4). Los dos dispositivos son un teclado externo en solitario y un reloj de tiempo real. El reloj se emplea para contar segundos, que sirven a otras partes del sistema (software).

Para establecer correctamente el vector de interrupciones para el nivel 4 (en la dirección $70), necesitamos inicializarlo cuando se inicie por primera vez el programa, como en:

```
MOVE.L   #EXTDEVS,$70   establ. vector nivel 4
```

Es obvio que puede haber otros registros por inicializar, en particular la pila del usuario y la pila de sistema. Por ejemplo, podemos usar:

```
MOVE.L   #STACK,SP      establ. pila sistema
MOVE.L   #USERTACK,A0   y pila usuario
```

donde establecemos A7 con la dirección STACK y A0 con la dirección USERSTACK.

La rutina de servicio de interrupciones EXTDEVS sería, por ejemplo:

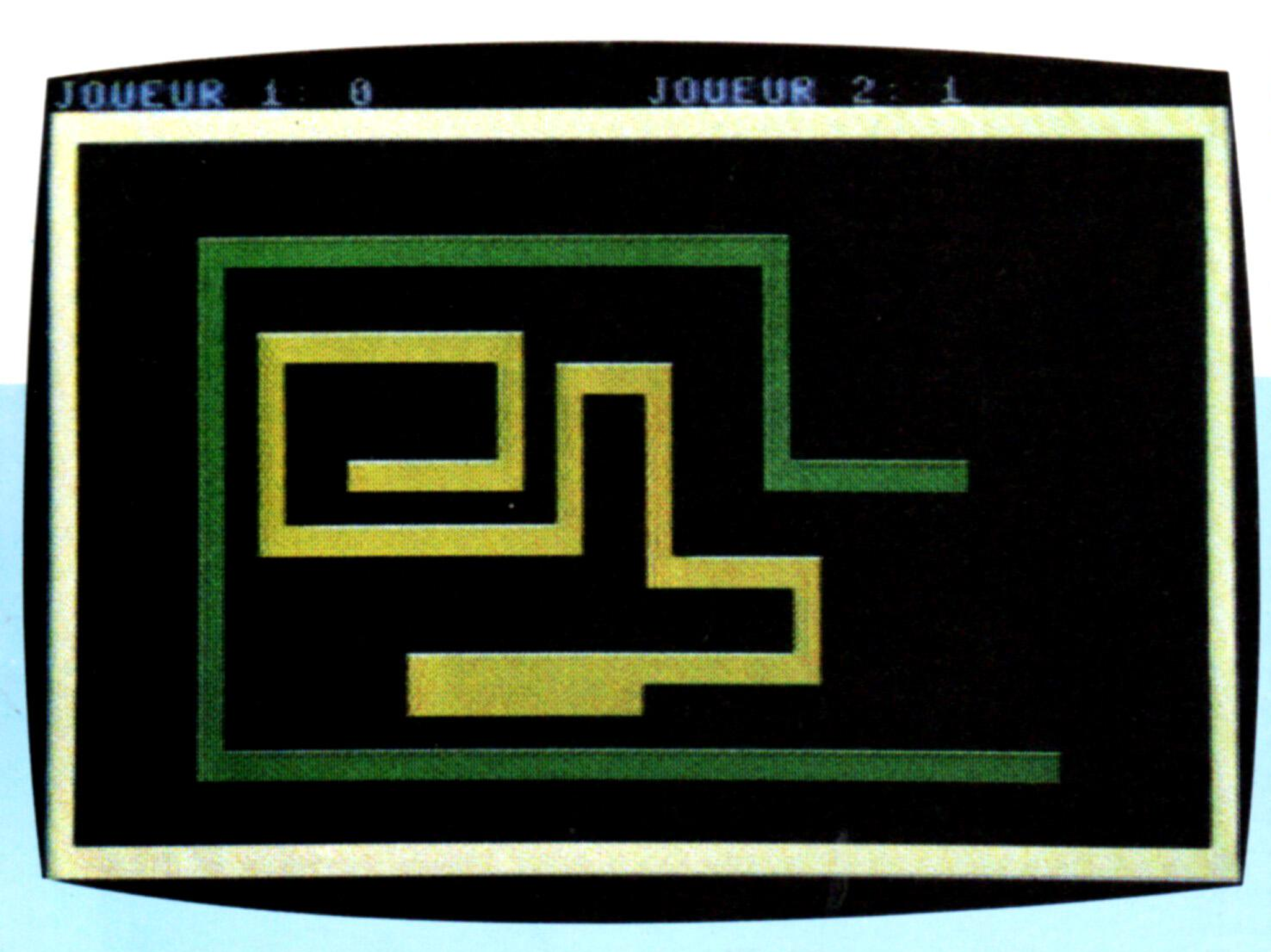
```
EXTDEVS  MOVEM.L  D0-D7,(A0) guarda los
                             regs. de datos
         BTST     #7,PIACRA  mira si está
                             el teclado
                             interrumpido
         BNE      CHAR
CKCLK    BTST     #6,PIACRB  si no, lo estará
                             el reloj
         BNE      CLOCK
         BRA      WILD       si no, entonces
                             espurio
CHAR     JSR      KEYBOARD   entra el
                             servicio
                             del teclado
         BRA      CKCLK      pero el reloj
                             puede estar
                             también
                             interrumpido!
CLOCK    JSR      SECS       entra el
                             servicio del reloj
WILD
         MOVEM.L  (SP)+,D0-  restaura los
                  D7         registros
                             de datos
         RTE
```

En esta rutina son guardados todos los registros de datos porque tanto CLOCK como CHAR los usarán. Nótese también que hacemos *pooling* de los dos dispositivos mirando los bits de estado de los registros en los registros de control del PIA. Comprobar el estado del reloj después de haber atendido al teclado, ya que puede tener a la vez dos interrupciones.

En el capítulo final de esta serie examinaremos el ensamblador y las facilidades que ofrece. Si se utilizan bien, quizá resulte la herramienta de desarrollo más poderosa de que dispone el programador.

Trazos en el C64

Como es sabido, este juego propone una auténtica "guerra de líneas". Pero los usuarios de un Commodore 64 ya pueden reemplazar la hoja de papel por la pantalla de su ordenador



Dos jugadores se enfrentan para dividirse el espacio vital. Cada uno de ellos debe esforzarse, al irse desplazando, para no cortar jamás su trazado o el de su adversario, y sin salirse del rectángulo dibujado en la pantalla. Los mandos a utilizar son:

Jugador de la derecha: <P>, <L>, <:> y <.>
Jugador de la izquierda: <W>, <A>, <S> y <Z>.

```
5 REM *************
10 REM *  TRAZOS  *
15 REM *************
20 GOSUB 1000
100 GET X$
110 C1=(X$="A")-(X$="S")+40*((X$="W")
    -(X$="Z"))
120 C2=(X$="L")-(X$=":")+40*((X$="P")
    -(X$="."))
130 IF C1<>0 THEN D1=C1
140 IF C2<>0 THEN D2=C2
150 P1=P1+D1
160 IF PEEK <P1><>32 THEN 3000
170 POKE P1,C
180 POKE P1+M,K1
190 P2=P2+D2
200 IF PEEK<P2><>32 THEN 4000
210 POKE P2,C
220 POKE P2+M,K2
230 FOR I=1 TO 50
240 NEXT I
250 GOTO 100
1000 P1=1514
1010 P2=1534
1020 K1=7
1030 K2=5
1040 C=160
1050 M=54272
1060 D1=1
1070 D2=-1
2000 PRINT CHR$<147>;
2010 POKE 53280,0
2020 POKE 53281,0
2030 FOR I=0 TO 39
2040 POKE 1064+I,C
2050 POKE 1064+I+M,1
2060 POKE 1984+I,C
2070 POKE 1984+I+M,1
2080 NEXT I
2090 FOR I=1 TO 22
2100 POKE 1064+I*40,C
2110 POKE 1064+I*40+M,1
2120 POKE 1103+I*40,C
2130 POKE 1103+I*40+M,1
2140 NEXT I
2160 POKE P1,C
2170 POKE P1+M,K1
2180 POKE P2,C
2190 POKE P2+M,K2
2200 PRINT "JUGADOR(1SPC)1:";J1,"JUGADOR
     (1SPC)2:";J2;
2210 RETURN
3000 J2=J2+1
3010 FOR I=1 TO 500
3020 GET X$
3030 NEXT I
3040 IF J2=10 THEN 5000
3050 GOTO 20
4000 J1=J1+1
4010 FOR I=1 TO 500
4020 GET X$
4030 NEXT I
4040 IF J1=10 THEN 5000
4050 GOTO 20
5000 PRINT CHR$(147);
5010 FOR I=1 TO 18
5020 PRINT
5030 NEXT I
5040 PRINT TAB(10)"EL(1SPC)JUGADOR
     (1SPC)";
5050 IF J1>J2 THEN PRINT "1";
5060 IF J2>J1 THEN PRINT "2";
5070 PRINT "(1SPC)GANA(1SPC)!"
5080 PRINT
5090 PRINT TAB(13)J1;"A";J2
5100 FOR I=1 TO 4
5110 PRINT
5120 GET X$
5130 NEXT I
5140 PRINT TAB(13)"OTRA(1SPC)?"
5150 GET X$
5160 IF X$="" THEN 5150
5170 IF X$<>"N" THEN RUN
5180 END
```

Editorial Delta, S.A.



10118

9 788485 822836